Research Report 217

Objects in the π-calculus

David Walker

RR217

Two semantics for a parallel object-oriented programming language are presented. One is a two-level transitional semantics in which the global behaviour of a system is derived directly from the possible actions of its constituent objects. The other is by translation into the π-calculus. A close correspondence between the semantics is established.

Department of Computer Science
University of Warwick
Coventry CV4 7AL
United Kingdom

April 1992

Objects in the π-calculus

David Walker
Department of Computer Science
University of Warwick

Abstract

Two semantics for a parallel object-oriented programming language are presented. One is a two-level transitional semantics in which the global behaviour of a system is derived directly from the possible actions of its constituent objects. The other is by translation into the π-calculus. A close correspondence between the semantics is established.

1 Introduction

The λ-calculus plays a fundamental rôle in computation theory. By encapsulating in undiluted form the paradigm of "computational entity as function" it provides both the definitive arena for the study of the paradigm and a foundation for its development and utilization. The discovery by Scott of models for the λ-calculus and its adoption by him and Strachey as a semantic basis for programming languages sparked the development of an elegant semantic theory which accommodates successfully the rich variety of sequential languages.

Efforts to extend the Scott-Strachey approach in an elegant way to concurrent languages were unsuccessful. This failure led Milner to propose an alternative conceptual framework for understanding concurrency and communication, a framework which found expression first in his calculus of communicating systems CCS [6] and subsequently in myriad process algebras. The radical idea of CCS was to view computational entities as agents whose *interaction* is the basic unit of behaviour. Each agent has a characteristic pattern of interaction, and by capturing structure algebraically the behaviour of a system of agents can be understood in terms of that of its components.

The function paradigm can be viewed as a special case of the agent paradigm in which interaction is of a restricted kind, namely communication between functions of arguments and results. This suggests that a fundamental calculus of communicating systems should have the λ-calculus embedded in it in some precise sense. The π-calculus [10, 11], which emerged as an elaboration and refinement of CCS, is a step towards such a fundamental calculus. Its primitive notions are two: *name* and *agent*. Names provide access to agents. More precisely, two agents which share a

name may interact by using it; and by mentioning names in interactions, agents may communicate to one another the ability to interact with other agents.

The ubiquity of naming is commonplace, not least in computer science. The novelty of the π-calculus lies in the attempt to found a tractable general theory of computation on the notion. The theoretical development and applications described in [10, 11] and subsequent papers, and in particular Milner's demonstration in [8] that the λ-calculus can be faithfully encoded in the π-calculus, suggest that the π-calculus has an important rôle to play in the search for more abstract theories of concurrent and communicating systems.

This paper seeks to elaborate this theme further by examining in detail how the π-calculus can be used as a semantic basis for a parallel object-oriented programming language. Providing adequate semantics for such languages is challenging. Among the most successful work is that on the POOL family of languages [3]. The starting point for the present paper is an operational semantics for a member of the family [1]. This was later complemented [2] by a denotational semantics based on metric spaces and a correspondence between the two semantics established [12]. Another semantics, using process algebra, was given in [13]. The nature of the language itself suggests that adequate models will be complicated and [2] provides confirmation of this. In particular, a clear view of the central concept, the object, is hard to discern although [5] makes further progress.

This paper offers a semantics for a slight variant of the POOL of [1] via a phrase by phrase translation into the π-calculus. Each syntactic entity is represented as an agent, the representations of composite entities being constructed simply from those of their constituents using the operators of the calculus. In particular, an object is represented as an agent whose algebraic structure reflects the internal structure of the object, and whose pattern of interaction captures the way in which the object may interact with others. An important contribution to the perspicuity of the translation is made by the use of a sort discipline in the π-calculus [9]. The translation modifies and extends techniques used in [6, 7] to give a semantics for a parallel programming language with shared variables by translation into CCS.

To relate the semantics offered here to the existing semantics, a correspondence is established between it and a reformulation of the operational semantics of [1]. The latter is given in terms of a transition system whose configurations represent global states of a system of objects. It is felt that there is independent interest in the reformulation in which rather than defining transitions between global configurations directly, as in [1], a labelled transition system describing the possible actions of objects in isolation is given and the global, interleaving transition relation defined in terms of this. Incidentally, this separation facilitates the definition on a common basis of other global transition relations, for example the *maximal parallelism* considered towards the end of [1].

The translation of POOL into the π-calculus extends to a mapping $[\![\cdot]\!]$ from global configurations Ω to agents. To state the correspondence let $\longrightarrow$ be the transition relation between global configurations, $\longrightarrow^*$ its reflexive and transitive closure,

$\Longrightarrow$ the reflexive and transitive closure of the $\xrightarrow{\tau}$ transition relation on agents, and $\sim$ the relation of strong bisimilarity on agents from [10, 11]. Let Ω_0 be the initial configuration associated with a POOL program, or *unit*. Then:

1. Any computation $\Omega_0 \longrightarrow \Omega_1 \longrightarrow \ldots \longrightarrow \Omega_n$ is directly mirrored by a derivation $[\![\Omega_0]\!] \Longrightarrow\sim [\![\Omega_1]\!] \Longrightarrow\sim \ldots \Longrightarrow\sim [\![\Omega_n]\!]$ which, because of the finer grain of action, typically involves more transitions.

2. If $[\![\Omega_0]\!] \Longrightarrow P$ then for some Ω, $\Omega_0 \longrightarrow^* \Omega$ and $P \Longrightarrow\sim [\![\Omega]\!]$.

A derivation from $[\![\Omega_0]\!]$ will typically involve many agents which are not translations of global configurations. Indeed there may be infinite derivations from $[\![\Omega_0]\!]$ in which no agent but the first represents a configuration. Such derivations can arise from delayed completion or non-completion of sequences of $\xrightarrow{\tau}$ transitions corresponding to single $\longrightarrow$ transitions. But any finite derivation can be continued so as to reflect a computation at the higher level. Indeed as the proof of 2 shows, a continuation $P \Longrightarrow\sim [\![\Omega]\!]$ can be found in which each of the agents representing an object in P completes a sequence of the kind mentioned above.

This paper treats a slight variant of the POOL of [1]. As discussed in the text, the language features not considered can be accommodated by extending the techniques presented. Inheritance and subtyping are absent from the POOL of [1] though see [4]. It is hoped that further developments along the lines examined here will contribute to improved theoretical understanding of these concepts. It is hoped also that through the development of model theory for the π-calculus, the semantics and correspondence result presented here will provide stepping-stones to a more abstract account of the meaning of the language.

The version of the π-calculus used in the translation differs somewhat from that of [10, 11]. It contains the composition and restriction operators, a generalized form of prefixing introduced in [9], and replication, introduced in [8], rather than full recursion. More is said on the π-calculus in the following section. The subsequent sections describe the language, the operational semantics, the sort discipline, the translation, and the correspondence between the semantics.

2 The π-calculus

The version of the π-calculus used here differs somewhat from that appearing in the original papers [10, 11]. With $x, y, \ldots$ ranging over an infinite set of *names*, *agents* $P, Q, \ldots$ are given as follows:

P	::=	$\mathbf{0}$	(inaction)
	|	$\pi.P$	(prefix)
	|	$P \mid Q$	(composition)
	|	$\langle y \rangle P$	(restriction)
	|	$!P$	(replication)

where prefixes π are given by:

$$\pi \;::=\; \overline{x}(y_1,\ldots,y_n) \;\mid\; x(y_1,\ldots,y_n)$$

where $n \geq 0$.

The more general form of prefix was introduced in [9], while replication appeared in [8]. Restriction of name y in agent P is written $\langle y\rangle P$ rather than $(y)P$ as in [10, 11]. For discussion of the intended interpretation of agents and missing definitions we refer to the papers cited. We write $\mathrm{fn}(P)$ for the set of free names, $\mathrm{bn}(P)$ for the set of bound names, and $\mathrm{n}(P)$ for the set of names of P, and similarly for actions α introduced below. We write also $P \equiv Q$ if P and Q are *structurally congruent*, this being the least relation such that:

1. Alpha-convertible agents are structurally congruent,
2. $P \mid Q \equiv Q \mid P, \quad P \mid (Q \mid R) \equiv (P \mid Q) \mid R, \quad P \mid \mathbf{0} \equiv P,$
3. $\langle x\rangle\langle y\rangle P \equiv \langle y\rangle\langle x\rangle P, \quad \langle x\rangle\mathbf{0} \equiv \mathbf{0},$
4. $\langle x\rangle(P \mid Q) \equiv P \mid \langle x\rangle Q \quad$ if $x \notin \mathrm{fn}(P)$,
5. $\langle x\rangle\pi.P \equiv \mathbf{0} \quad$ if $\pi = \overline{x}(\tilde{y})$ or $x(\tilde{y})$, and
6. $!P \equiv P \mid !P.$

We treat structurally-congruent agents as identical, and for $\tilde{y}$ a sequence $y_1,\ldots,y_n$ of names we write $\langle\tilde{y}\rangle P$ for $\langle y_1\rangle\ldots\langle y_n\rangle P$. We also abbreviate an agent such as $\langle y_1,y_2,y_3\rangle\overline{x}(z_1,y_2,y_3,z_2).P$ to $\langle y_1\rangle\overline{x}(z_1,\langle y_2,y_3\rangle,z_2).P$. Finally, we write π for $\pi.\mathbf{0}$.

The actions α labelling the arrows in the transition relations are given by:

$$\alpha \;::=\; \overline{x}(\langle\tilde{z}\rangle\tilde{y}) \;\mid\; x(\langle\tilde{z}\rangle\tilde{y}) \;\mid\; \tau$$

where in $\overline{x}(\langle\tilde{z}\rangle\tilde{y})$ and $x(\langle\tilde{z}\rangle\tilde{y})$, $\tilde{z}$ is a subset of the names occurring in the sequence $\tilde{y}$; these are the bound names of the action. Recalling that structurally-congruent agents are regarded as identical, the transition rules are as follows:

(IN) $\quad x(\tilde{y}).P \xrightarrow{x(\langle\tilde{z}\rangle\tilde{w})} P\{\tilde{w}/\tilde{y}\}$, provided $\tilde{z} \cap \mathrm{fn}(\langle\tilde{y}\rangle P) = \emptyset$

(OUT) $\quad \overline{x}(\tilde{y}).P \xrightarrow{\overline{x}(\tilde{y})} P$

(PAR) $\quad$ If $P \xrightarrow{\alpha} P'$ and $\mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$ then $P \mid Q \xrightarrow{\alpha} P' \mid Q$

(COM) $\quad$ If $P \xrightarrow{\overline{x}(\langle\tilde{z}\rangle\tilde{y})} P'$ and $Q \xrightarrow{x(\langle\tilde{z}\rangle\tilde{y})} Q'$ then $P \mid Q \xrightarrow{\tau} \langle\tilde{z}\rangle(P' \mid Q')$

(RES) $\quad$ If $P \xrightarrow{\alpha} P'$ and $y \notin \mathrm{n}(\alpha)$ then $\langle y\rangle P \xrightarrow{\alpha} \langle y\rangle P'$

(OPEN) $\quad$ If $P \xrightarrow{\overline{x}(\langle\tilde{z}\rangle\tilde{y})} P'$, $w \neq x$, $w \in \tilde{y}$ and $w \notin \tilde{z}$, then $\langle w\rangle P \xrightarrow{\overline{x}(\langle\tilde{z},w\rangle\tilde{y})} P'$

We write $\sim$ for the relation of strong bisimilarity of agents defined by generalizing the definition of [11] to the present setting.

The translation employs a sort discipline of a kind introduced in [9]. Let Σ be a set of *subject sorts* and Σ^* the corresponding set of *object sorts*. A *sorting* is a mapping $\sigma : \Sigma \longrightarrow \Sigma^*$. We write $x : S \longrightarrow (S_1, \ldots, S_n)$ to express that S is a subject sort with $(S_1, \ldots, S_n)$ its associated object sort and x a name of sort S. In an agent which conforms to the sort discipline given by σ, any occurrence of x or $\overline{x}$ as a subject must have an object $(y_1, \ldots, y_n)$ with $y_1 : S_1, \ldots, y_n : S_n$. As an example suppose $\Sigma = \{S, T, U\}$ and σ is given by:

$$\begin{array}{rcl} x, x' & : & S \longrightarrow (T, U) \\ y & : & T \longrightarrow (S) \\ z, z' & : & U \longrightarrow (S, U) \end{array}$$

Then $x(y,z).\overline{y}\langle x'\rangle.\overline{z}(x',z).\mathbf{0}$ and $y(x).\overline{x}(y,\langle z\rangle).z(x',z').\mathbf{0}$ conform to σ.

3 The Language

Characteristic of the object-oriented style of programming is the description of a computational system as a collection of *objects* each of which is a self-contained entity possessing data and procedures, or *methods*. Objects interact by sending messages to one another. A message may be either a request by the sender for the receiver to execute one of its methods with parameters, references to objects, provided by the sender, or a reply, again a reference, from the receiver to the sender in response to such a method invocation. The communication structure of a system may thus change as computation proceeds. Furthermore, objects may be created during computation.

In the variant of POOL considered here a program, or *unit*, consists of a sequence of *class declarations* together with an indication of which class furnishes the *root object* which alone exists at the beginning of a computation of the unit. A class declaration consists of sequences of variable declarations and method declarations together with a statement, the *body* of the class. Each object is an *instance* of a class and on creation executes a replica of the body of the class in parallel with all other existing objects using its own replicas of the variables and methods of the class. Statements are constructed by sequencing, conditional and iterative operators from expressions, assignment statements and the answer statement by means of which an object accepts a request to execute one of its methods.

Among the forms of expression are: new(B), whose evaluation creates a new object of class B and returns a reference to that object; self, whose evaluation returns, roughly speaking, a reference to the object in which it occurs; and $E!M(E_1, \ldots, E_n)$, whose evaluation involves left-to-right evaluation of the expressions $E, E_1, \ldots, E_n$ and the sending to the object to which the value of E is a reference of a request to execute its method M with parameters the references which are the values of

$E_1, \ldots, E_n$. The value of $E!M(E_1, \ldots, E_n)$ is the reference returned by the object in which the method is invoked, the activity of the sender being suspended until this is received.

The syntax of the language is as follows, where A, B, C range over class names, M over method names, X, Y over variables, E over expressions, S over statements, and U over units. First, expressions are given as follows:

$$
\begin{array}{lcll}
E & ::= & \mathsf{b} & (\mathsf{b} = \mathsf{true}, \mathsf{false}) \\
 & | & \mathsf{new}(B) & (B \neq \mathsf{Bool}) \\
 & | & \mathsf{self} & \\
 & | & X & \\
 & | & E!M(E_1, \ldots, E_n) & \\
 & | & M(E_1, \ldots, E_n) & \\
 & | & \mathit{Vdecs}\ \mathsf{in}\ E & \\
 & | & S; E &
\end{array}
$$

Bool is the sole *standard class*, and true and false expressions whose values are references to the *standard objects* of that class. $M(E_1, \ldots, E_n)$ is a local call of the method M. *Vdecs* is a sequence of variable declarations:

$$\mathit{Vdecs} \ ::= \ \mathsf{var}\ X_1 : A_1\ , \ldots,\ X_n : A_n$$

Statements are given by:

$$
\begin{array}{lcl}
S & ::= & X := E \\
 & | & \mathsf{answer} \\
 & | & E \\
 & | & S_1; S_2 \\
 & | & \mathsf{if}\ E\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2 \\
 & | & \mathsf{while}\ E\ \mathsf{do}\ S
\end{array}
$$

Note that an expression is a statement.

Method declarations are given by:

$$\mathit{Mdec} \ ::= \ \mathsf{method}\ M(Y_1 : C_1\ , \ldots,\ Y_p : C_p) : C\ \mathsf{is}\ E$$

with C the result class of the method, $Y_1, \ldots, Y_p$ the formal parameters, and E the body. Note that the expression form $S; E$ allows, among other things, answer statements to appear in method bodies, and hence nested answer statements.

Sequences of method declarations are given by:

$$\mathit{Mdecs} \ ::= \ \mathit{Mdec}_1\ , \ldots,\ \mathit{Mdec}_q$$

class declarations by:

$$\mathit{Cdec} \ ::= \ \mathsf{class}\ A\ \mathsf{is}\ \mathit{Vdecs}, \mathit{Mdecs}\ \mathsf{with\ body}\ S$$

and unit declarations by:

$$\mathit{Udec} \ ::= \ \mathsf{unit}\ U\ \mathsf{is}\ \mathit{Cdec}_1\ , \ldots,\ \mathit{Cdec}_r\ \mathsf{with\ root}\ A$$

The language is strongly typed and various straightforward non-context-free conditions are imposed, e.g. that in *Udec* above, 'A' must be the name of one of the classes declared in $Cdec_1, \ldots, Cdec_r$, and that if an expression $E!M(E_1, \ldots, E_n)$ appears in one of the class definitions and the type of E is C, then there must be a class named 'C' and it must have a method named 'M' of the appropriate type. We consider only units in which all such conditions are met; see [1] for full details.

The language differs from the POOL of [1] in some minor syntactic details. Significant differences are that in [1]: an answer statement may proscribe certain methods, an intricate select statement allowing a conditional answer of a message is present, variables are separated into *instance* variables and *local* variables, and a standard class *integer* is present; here only the simpler answer statement which allows any method to be invoked is considered, the select statement is omitted, and local variable declarations are allowed in expressions. A class representing integers can be defined without undue difficulty. The π-calculus translation can be extended easily to handle the more general form of answer statement and, slightly less easily, the select statement.

4 Operational Semantics

The transitional semantics of [1] is reformulated by defining a labelled transition system which describes the possible actions of objects in isolation and from this defining the global transition relation. We assume a set of *object names* ranged over by α, β, γ which in addition to a distinguished name nil and names tt and ff for the standard objects, contains infinitely-many other names. We refer to nil, tt and ff as the *standard names*. The following kinds of entity are used in the transitional semantics.

A *family of method definitions* is a finite partial function μ from method names to pairs $\langle \tilde{Y}, E \rangle$, with $\tilde{Y}$ the sequence of formal parameters, and E the body of the associated method.

A *class definition* is a triple $\langle \tilde{X}, \mu, S \rangle$, with $\tilde{X}$ the sequence of variables, μ the family of method definitions, and S the body of the associated class.

An *environment* is a pair $\Phi = \langle \Phi_1, \Phi_2 \rangle$ with Φ_1 a finite partial function from class names to class definitions, and Φ_2 a finite partial function from object names to class names.

A *store* is a finite partial map σ, ρ from variables to object names.

A *state* is a finite set Γ of triples $\langle \alpha, S, \sigma \rangle$.

Finally, a *configuration* is a pair $\langle \Phi, \Gamma \rangle$ consisting of an environment and a state.

Briefly, a configuration $\langle \Phi, \Gamma \rangle$ provides an instantaneous description of a computation of a unit as follows. Φ_1 records the definitions of the classes of the unit and is unchanged throughout the computation. Φ_2 records the name and class of each

existing object; it grows each time an object is created. Γ records the local store and the yet-to-be-executed part of the body of each existing object. In the initial configuration, Φ_2 will be of the form $\{\langle \alpha, A\rangle\}$ where A is the class of the root object α, and Γ of the form $\{\langle \alpha, S, \sigma\rangle\}$ where $\Phi_1(A) = \langle \tilde{X}, \mu, S\rangle$ and, as a variable initially contains a reference to no object, $\sigma = \{\tilde{X} := \mathsf{nil}\}$. As described later, any reachable configuration will satisfy certain coherence conditions.

To give the transitional semantics we augment the class of expressions as follows:

$$\begin{array}{rcl} E & ::= & \ldots \\ & | & \beta \\ & | & \mathsf{wait}(\beta) \\ & | & E \Rightarrow \beta \\ & | & (E, \rho) \end{array}$$

The rôles of these expressions will be explained below.

The global transition relation on configurations

$$\langle \Phi, \Gamma\rangle \longrightarrow \langle \Phi', \Gamma'\rangle$$

is defined in terms of a family of local transition relations

$$\langle \alpha, \mu\rangle \vdash \langle S, \sigma\rangle \stackrel{\ell}{\longmapsto} \langle S', \sigma'\rangle$$

where the label ℓ is either empty or of one of the forms $\mathrm{new}(B, \beta)$, $\mathrm{send}(\beta, M, \tilde{\beta})$, $\mathrm{answer}(\beta, M, \tilde{\beta})$, $\mathrm{result}(\beta, \gamma)$ or $\mathrm{return}(\beta, \gamma)$.

4.1 The Local Transition Relations

The local relations are defined by the following axioms and rules.

Axioms

First we have the axioms for expressions.

(TRUE) $\quad \langle \alpha, \mu\rangle \vdash \langle \mathsf{true}, \sigma\rangle \longmapsto \langle \mathsf{tt}, \sigma\rangle$

(FALSE) $\quad \langle \alpha, \mu\rangle \vdash \langle \mathsf{false}, \sigma\rangle \longmapsto \langle \mathsf{ff}, \sigma\rangle$

(NEW) $\quad \langle \alpha, \mu\rangle \vdash \langle \mathsf{new}(B), \sigma\rangle \stackrel{\mathrm{new}(B,\beta)}{\longmapsto} \langle \beta, \sigma\rangle$

provided β is not a standard name

(SELF) $\quad \langle \alpha, \mu\rangle \vdash \langle \mathsf{self}, \sigma\rangle \longmapsto \langle \alpha, \sigma\rangle$

(VAR) $\quad \langle \alpha, \mu\rangle \vdash \langle X, \sigma\rangle \longmapsto \langle \beta, \sigma\rangle$

where $\beta = \sigma(X)$

(SEND1) $\langle \alpha, \mu \rangle \vdash \langle \beta ! M(\tilde{\beta}), \sigma \rangle \overset{\mathsf{send}(\beta, M, \tilde{\beta})}{\longmapsto} \langle \mathsf{wait}(\beta), \sigma \rangle$

provided β is not a standard name

(CALL1) $\langle \alpha, \mu \rangle \vdash \langle M(\tilde{\beta}), \sigma \rangle \longmapsto \langle (E, \rho), \sigma \rangle$

where $\mu(M) = \langle \tilde{Y}, E \rangle$ and $\rho = \{\tilde{Y} := \tilde{\beta}\}$

(VARDEC) $\langle \alpha, \mu \rangle \vdash \langle \mathit{Vdecs}\ \mathsf{in}\ E, \sigma \rangle \longmapsto \langle (E, \rho), \sigma \rangle$

where $\mathit{Vdecs} = \mathsf{var}\ \tilde{Y}$ and $\rho = \{\tilde{Y} := \mathsf{nil}\}$

In (TRUE) and (FALSE), true and false evaluate to the names tt and ff respectively. In (NEW), the name β may be chosen arbitrarily from among the non-standard names; a side-condition in one of the global rules will ensure that a fresh name is used. (SELF) and (VAR) are straightforward. In (VAR), if $\sigma(X) = \mathsf{nil}$ then X contains a reference to no object. (SEND1) may be applied when all subexpressions have been evaluated; in the global transition system a 'send' action will synchronize with an 'answer' action emanating from the axiom (ANSWER) below. The auxiliary expression $\mathsf{wait}(\beta)$ captures the suspension of the activity of α until β returns an answer; see the axiom (WAIT) below. To reflect the view that the attempted evaluation of an expression $\mathsf{nil}!M(E_1, \ldots, E_n)$ may result in an error we might add an 'error' configuration and suitable rules. The semantics here will instead not allow the object containing such an expression to proceed. In (CALL1) the appropriate method body E, obtained by referring to μ, is evaluated with respect to an augmented local store, represented by the auxiliary expression (E, ρ), in which are recorded the actual parameters. Finally, (VARDEC) is similar but with the local variables assigned nil. There are no axioms for names, while $S; E$ is catered for by the axiom (SEQ1) and the rule (SEQ2) below.

The axioms for statements are as follows.

(ASSIGN1) $\langle \alpha, \mu \rangle \vdash \langle X := \beta, \sigma \rangle \longmapsto \langle \mathsf{nil}, \sigma' \rangle$

where $\sigma' = \sigma[X := \beta]$

(ANSWER) $\langle \alpha, \mu \rangle \vdash \langle \mathsf{answer}, \sigma \rangle \overset{\mathsf{answer}(\beta, M, \tilde{\beta})}{\longmapsto} \langle (E, \rho) \Rightarrow \beta, \sigma \rangle$

where $\mu(M) = \langle \tilde{Y}, E \rangle$ and $\rho = \{\tilde{Y} := \tilde{\beta}\}$,
and provided β is not a standard name

(SEQ1) $\langle \alpha, \mu \rangle \vdash \langle \beta; S, \sigma \rangle \longmapsto \langle S, \sigma \rangle$

(COND1) $\langle \alpha, \mu \rangle \vdash \langle \mathsf{if\ tt\ then}\ S_1\ \mathsf{else}\ S_2, \sigma \rangle \longmapsto \langle S_1, \sigma \rangle$

(COND2) $\langle \alpha, \mu \rangle \vdash \langle \mathsf{if\ ff\ then}\ S_1\ \mathsf{else}\ S_2, \sigma \rangle \longmapsto \langle S_2, \sigma \rangle$

(LOOP) $\langle \alpha, \mu \rangle \vdash \langle \mathsf{while}\ E\ \mathsf{do}\ S, \sigma \rangle \longmapsto \langle \mathsf{if}\ E\ \mathsf{then}\ (S; \mathsf{while}\ E\ \mathsf{do}\ S)\ \mathsf{else\ nil}, \sigma \rangle$

In (ASSIGN1) the store is updated with the appropriate value. In (ANSWER) a request from any object to execute any appropriate method with any appropriate parameters may be accepted, with $(E, \rho) \Rightarrow \beta$ reflecting that the appropriate body should be evaluated with the given actual parameters held in an augmented store and the result returned to the sender; see (RETURN1) and (RETURN2) below. The other axioms are straightforward.

With $\mathsf{b}, \mathsf{b}' = \mathsf{tt}, \mathsf{ff}$, the axioms for the methods of the standard class Bool are as follows.

(NOT) $\langle \alpha, \mu \rangle \vdash \langle \mathsf{b!\ not}\ (), \sigma \rangle \longmapsto \langle \mathsf{b}', \sigma \rangle$ where $b' = \neg b$

(AND) $\langle \alpha, \mu \rangle \vdash \langle \mathsf{b!\ and}\ (\mathsf{b}'), \sigma \rangle \longmapsto \langle \mathsf{b}'', \sigma \rangle$ where $b'' = b \wedge b'$

The final axioms concern the auxiliary expressions.

(WAIT) $\langle \alpha, \mu \rangle \vdash \langle \mathsf{wait}(\beta), \sigma \rangle \overset{\mathsf{result}(\beta,\gamma)}{\longmapsto} \langle \gamma, \sigma \rangle$

(RETURN1) $\langle \alpha, \mu \rangle \vdash \langle (\gamma, \rho), \sigma \rangle \longmapsto \langle \gamma, \sigma \rangle$

(RETURN2) $\langle \alpha, \mu \rangle \vdash \langle \gamma \Rightarrow \beta, \sigma \rangle \overset{\mathsf{return}(\beta,\gamma)}{\longmapsto} \langle \mathsf{nil}, \sigma \rangle$

(WAIT) captures the reactivation of α when β returns any result γ to it. The complementary (RETURN1) and (RETURN2) describe how an evaluated expression is returned. In the global transition system, 'result' and 'return' actions will synchronize.

Rules

To complete the definition of the local transition relations we have the following eight rules. The first six share the premise:

$$\langle \alpha, \mu \rangle \vdash \langle E, \sigma \rangle \overset{\ell}{\longmapsto} \langle E', \sigma' \rangle$$

The conclusions of the rules are:

(SEND2) $\langle \alpha, \mu \rangle \vdash \langle E!M(\tilde{E}), \sigma \rangle \overset{\ell}{\longmapsto} \langle E'!M(\tilde{E}), \sigma' \rangle$

(SEND3) $\langle \alpha, \mu \rangle \vdash \langle \beta!M(\tilde{\beta}, E, \tilde{E}), \sigma \rangle \overset{\ell}{\longmapsto} \langle \beta!M(\tilde{\beta}, E', \tilde{E}), \sigma' \rangle$

(CALL2) $\langle \alpha, \mu \rangle \vdash \langle M(\tilde{\beta}, E, \tilde{E}), \sigma \rangle \overset{\ell}{\longmapsto} \langle M(\tilde{\beta}, E', \tilde{E}), \sigma' \rangle$

(RETURN3) $\langle \alpha, \mu \rangle \vdash \langle E \Rightarrow \beta, \sigma \rangle \overset{\ell}{\longmapsto} \langle E' \Rightarrow \beta, \sigma' \rangle$

(ASSIGN2) $\langle \alpha, \mu \rangle \vdash \langle X := E, \sigma \rangle \overset{\ell}{\longmapsto} \langle X := E', \sigma' \rangle$

(COND3) $\langle \alpha, \mu \rangle \vdash \langle \mathsf{if}\ E\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2, \sigma \rangle \overset{\ell}{\longmapsto} \langle \mathsf{if}\ E'\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2, \sigma' \rangle$

These rules are straightforward, the first three capturing left-to-right evaluation of parameters. The final two rules are:

(SEQ2) $$\frac{\langle \alpha, \mu \rangle \vdash \langle S_1, \sigma \rangle \overset{\ell}{\longmapsto} \langle S_1', \sigma' \rangle}{\langle \alpha, \mu \rangle \vdash \langle S_1; S_2, \sigma \rangle \overset{\ell}{\longmapsto} \langle S_1'; S_2, \sigma' \rangle}$$

(LOCAL) $$\frac{\langle \alpha, \mu \rangle \vdash \langle E, \sigma \oplus \rho \rangle \overset{\ell}{\longmapsto} \langle E', \tau \rangle}{\langle \alpha, \mu \rangle \vdash \langle (E, \rho), \sigma \rangle \overset{\ell}{\longmapsto} \langle (E', \rho'), \sigma' \cup \sigma'' \rangle}$$

where $\rho' = \tau\lceil \mathrm{dom}(\rho)$, $\sigma' = \tau\lceil(\mathrm{dom}(\sigma) - \mathrm{dom}(\rho))$
and $\sigma'' = \sigma\lceil(\mathrm{dom}(\sigma) \cap \mathrm{dom}(\rho))$

(SEQ2) is straightforward and (LOCAL) captures the effect of evaluation with respect to an augmented local store.

4.2 The Global Transition Relation

The global transition relation

$$\langle \Phi, \Gamma \rangle \longrightarrow \langle \Phi', \Gamma' \rangle$$

is defined from the local relations by the following four rules. They describe how the configuration may change as a result of the independent activity of one object (LOCAL), the creation of a new object (NEW), the initiation of a method invocation (COM1), and the return of a result of a method invocation (COM2). A coherence side condition common to all the rules is that if $\langle \alpha, \mu \rangle$ appears in a premise and Φ in a conclusion then $\mu = \Phi_1(\Phi_2(\alpha))_2$, i.e. the family of method definitions referred to is the appropriate one for an object of α's class.

(LOCAL) $$\frac{\langle \alpha, \mu \rangle \vdash \langle S, \sigma \rangle \longmapsto \langle S', \sigma' \rangle}{\langle \Phi, \Gamma \cup \{\langle \alpha, S, \sigma \rangle\} \rangle \longrightarrow \langle \Phi, \Gamma \cup \{\langle \alpha, S', \sigma' \rangle\} \rangle}$$

(NEW) $$\frac{\langle \alpha, \mu \rangle \vdash \langle S, \sigma \rangle \overset{\mathrm{new}(B,\beta)}{\longmapsto} \langle S', \sigma' \rangle}{\langle \Phi, \Gamma \cup \{\langle \alpha, S, \sigma \rangle\} \rangle \longrightarrow \langle \Phi', \Gamma \cup \{\langle \alpha, S', \sigma' \rangle, \langle \beta, S'', \sigma'' \rangle\} \rangle}$$

provided $\beta \notin \mathrm{names}(\Gamma) \cup \{\alpha\}$, $\Phi' = \langle \Phi_1, \Phi_2[\beta := B] \rangle$,
$\Phi_1(B) = \langle \tilde{X}, \mu'', S'' \rangle$, and $\sigma'' = \{\tilde{X} := \mathsf{nil}\}$

(COM1)

$$\frac{\langle \alpha, \mu_\alpha \rangle \vdash \langle S_\alpha, \sigma_\alpha \rangle \overset{\text{send}(\beta, M, \tilde{\beta})}{\longmapsto} \langle S'_\alpha, \sigma'_\alpha \rangle \quad \langle \beta, \mu_\beta \rangle \vdash \langle S_\beta, \sigma_\beta \rangle \overset{\text{answer}(\alpha, M, \tilde{\beta})}{\longmapsto} \langle S'_\beta, \sigma'_\beta \rangle}{\langle \Phi, \Gamma \cup \{\langle \alpha, S_\alpha, \sigma_\alpha \rangle, \langle \beta, S_\beta, \sigma_\beta \rangle\} \rangle \longrightarrow \langle \Phi, \Gamma \cup \{\langle \alpha, S'_\alpha, \sigma'_\alpha \rangle, \langle \beta, S'_\beta, \sigma'_\beta \rangle\} \rangle}$$

(COM2)

$$\frac{\langle \alpha, \mu_\alpha \rangle \vdash \langle S_\alpha, \sigma_\alpha \rangle \overset{\text{result}(\beta, \gamma)}{\longmapsto} \langle S'_\alpha, \sigma'_\alpha \rangle \quad \langle \beta, \mu_\beta \rangle \vdash \langle S_\beta, \sigma_\beta \rangle \overset{\text{return}(\alpha, \gamma)}{\longmapsto} \langle S'_\beta, \sigma'_\beta \rangle}{\langle \Phi, \Gamma \cup \{\langle \alpha, S_\alpha, \sigma_\alpha \rangle, \langle \beta, S_\beta, \sigma_\beta \rangle\} \rangle \longrightarrow \langle \Phi, \Gamma \cup \{\langle \alpha, S'_\alpha, \sigma'_\alpha \rangle, \langle \beta, S'_\beta, \sigma'_\beta \rangle\} \rangle}$$

The side condition on β in (NEW) ensures that a new object receives a fresh name; names(Γ) is the set of first components of the elements of Γ. The harmonization between sender and receiver in the rules (COM1) and (COM2) is achieved via the labels on the arrows in the premises.

The *initial configuration* associated with a unit declaration

$$Udec \equiv \text{unit } U \text{ is } Cdec_1, \ldots, Cdec_r \text{ with root } A$$

is $\langle \Phi, \Gamma \rangle$ where Φ_1 is given by the definitions of the classes, $\Phi_2 = \{\langle \alpha, A \rangle\}$ for an arbitrary α, and $\Gamma = \{\langle \alpha, S, \sigma \rangle\}$ where $S = \Phi_1(A)_3$ and $\sigma = \{\tilde{X} := \text{nil}\}$ where $\tilde{X} = \Phi_1(A)_1$.

5 The Sort Discipline

The perspicuity of the translation is much enhanced by the imposition of a sort discipline. Most of the sorts are parametrized on class names or sequences of class names. For example, for each class name A there are subject sorts CLASS[A] and OBJECT[A], and for each sequence $C_1, \ldots, C_p, C$ of class names a subject sort METHOD[$C_1, \ldots, C_p, C$]. In the sorting an expression

$$s : S \longrightarrow (S_1, \ldots, S_n)$$

indicates that S is a subject sort, $(S_1, \ldots, S_n)$ the associated object sort, and s a typical name of sort S.

The core of the sorting is:

$$\begin{array}{rlll} k: & \text{CLASS}[A] & \longrightarrow (\text{LINK}[A]) & (A \neq \text{BOOL}) \\ l: & \text{LINK}[A] & \longrightarrow (\text{OBJECT}[A]) & \\ a, c: & \text{OBJECT}[A] & \longrightarrow (\text{METHODS}[A]) & \end{array}$$

where if A has methods of types $C_{i1} \times \ldots \times C_{ip_i} \longrightarrow C_i$ for $i = 1, \ldots, q$ then

$$\tilde{m}: \ \mathsf{METHODS}[A] = \mathsf{METHOD}[C_{11}, \ldots, C_{1p_1}, C_1], \ldots, \mathsf{METHOD}[C_{q1}, \ldots, C_{qp_q}, C_q]$$
$$m: \ \mathsf{METHOD}[C_1, \ldots, C_p, C] \longrightarrow (\mathsf{OBJECT}[C_1], \ldots, \mathsf{OBJECT}[C_p], \mathsf{LINK}[C])$$

In addition the sorting contains:

$$x: \ \mathsf{VAR}[A] \longrightarrow (\mathsf{LINK}[A], \mathsf{LINK}[A])$$
$$l: \ \mathsf{NULL} \longrightarrow ()$$
$$g: \ \mathsf{GO}(C) \longrightarrow (\mathsf{LINK}[C])$$

For the standard class Bool:

$$bk: \ \mathsf{CLASS[BOOL]} \longrightarrow (\mathsf{LINK[BOOL]}, \mathsf{TRUE}, \mathsf{FALSE})$$
$$t: \ \mathsf{TRUE} \longrightarrow ()$$
$$f: \ \mathsf{FALSE} \longrightarrow ()$$
$$p: \ \mathsf{PAIR} \longrightarrow (\mathsf{TRUE}, \mathsf{FALSE})$$

and also

$$b: \mathsf{OBJECT[BOOL]} \longrightarrow (\mathsf{METHODS[BOOL]})$$
$$\tilde{m}: \mathsf{METHODS[BOOL]} = \mathsf{PAIR}, \mathsf{METHOD[BOOL]}, \mathsf{METHOD[BOOL, BOOL]}$$

where the methods are value, not and and.

The sorting itself conveys something of the abstract structure of the language. Rather than attempting to explain it fully, which would require anticipating the translation in detail, we focus briefly on its core. First consider $\mathsf{CLASS}[A] \longrightarrow (\mathsf{LINK}[A])$ and $\mathsf{LINK}[A] \longrightarrow (\mathsf{OBJECT}[A])$. A class declaration will be represented as a replicator which may provide an indefinite number of agents representing objects of that class. The replicator representing the class A will have a free name of sort $\mathsf{CLASS}[A]$ on which it may receive a name of sort $\mathsf{LINK}[A]$; it will then pass along that name a private name of sort $\mathsf{OBJECT}[A]$, that private name being a name free in an agent representing a new object of class A. The only agents which may interact with a replicator representing a class in this way will be those representing expressions of the form $\mathsf{new}(A)$.

Now consider the remainder of the core. Objects interact by sending messages, an interaction consisting in a request to execute a method with specified parameters. This is captured by the remaining clauses of the core in a way which is explained fully at the appropriate points in the translation. Briefly, an agent representing a sending object sends to an agent representing an answering object a sequence of private names of sorts corresponding to the methods of the class of the receiving object. It then indicates its interest in a particular method by sending on the private name associated with *that* method, names which are free in the agents representing the parameters of the call and a name of sort $\mathsf{LINK}[C]$ along which a name free in the agent representing the result of the call should be returned.

6 The Translation

Each syntactic entity is represented as an agent, the representations of composite entities being obtained simply from those of their constituents. There are many cross-links between different parts of the translation.

A statement S is translated as an agent $[\![S]\!](l, a, \tilde{x}, \tilde{m}, \tilde{k}, \tilde{c}, \tilde{l})$ where the parameters, some of which may be absent, are as follows:

1. If S is an expression of type C then l : LINK[C] is a name at which the value of S, or more precisely a name free in the agent representing the value, may be delivered. If S is not an expression then l : NULL is a name at which termination of execution of $[\![S]\!]$ may be signalled.
2. a is a name free in the agent representing the object in which S occurs; it corresponds to an "object identifier" – see particularly [[self]] and [[answer]] below.
3. $\tilde{x}$ are names free in agents representing variables occurring in S.
4. $\tilde{m}$ are names free in agents representing methods of the class of the object in which S occurs.
5. $\tilde{k}$ are names free in agents representing classes whose names occur in S.
6. $\tilde{c}$ are names free in agents representing objects to which parts of S contain references. Some of these names may represent references to no object.
7. $\tilde{l}$ are names of "return links" along which the object in which S occurs may receive or return results of method calls.

To avoid excessive notation a list of parameters as above will often be omitted or abbreviated to highlight significant points.

First in the translation we have four building blocks:

$$\text{Value}(l, c) \equiv \bar{l}(c)$$

$$\text{Null}(l) \equiv \bar{l}$$

$$\text{Wait}(l, l') \equiv l'(c).\,\text{Value}(l, c)$$

$$\text{Return}(l, l', l'') \equiv l''(c).\,\bar{l'}(c).\,\text{Null}(l)$$

Value represents an evaluated expression, c being a name free in the agent representing the object to which the value is a reference. Null represents a statement which is not an expression and whose execution is all but complete. Wait and Return will be used to model the passing of messages.

For the basic expressions of the standard class Bool:

$$[\![\mathsf{true}]\!](l, bk) \equiv \overline{bk}\langle l', t, f\rangle.\, \bar{t}.\, l'(b).\, \mathrm{Value}(l, b)$$

$$[\![\mathsf{false}]\!](l, bk) \equiv \overline{bk}\langle l', t, f\rangle.\, \bar{f}.\, l'(b).\, \mathrm{Value}(l, b)$$

It will be seen below that the standard class Bool is represented as a replicator BoolClass having a free name bk, and the standard objects as agents Bool_t and Bool_f. The agents $[\![\mathsf{true}]\!]$ and $[\![\mathsf{false}]\!]$ send to BoolClass three private names. The first is a return link while the other two are used to differentiate between the Boolean values using an idea appearing often in the translation: By passing the pair (t, f) of private names and then communicating on t, $[\![\mathsf{true}]\!]$ requests BoolClass to provide a link to the agent Bool_t representing the Boolean value *true*; $[\![\mathsf{false}]\!]$ acts similarly but communicates on f to request a link to the agent Bool_f representing *false*.

For the remaining simple expressions:

$$[\![\mathsf{new}(B)]\!](l, k) \equiv \bar{k}\langle l'\rangle.\, l'(c).\, \mathrm{Value}(l, c)$$

$$[\![\mathsf{self}]\!](l, a) \equiv \mathrm{Value}(l, a)$$

$$[\![X]\!](l, x) \equiv \bar{x}\langle r, u\rangle.\, r(c).\, \mathrm{Value}(l, c)$$

It is convenient to give here the clause for a variable declaration:

$$[\![\mathsf{var}\ X]\!](x, c) \equiv \mathrm{Var}(x, c)$$

$$\begin{aligned}\mathrm{Var}(x, c) \equiv\ & \langle l\rangle\ (x(r, u).\, (\bar{r}(c).\, \bar{l}(c) \mid u(c').\, \bar{l}(c')) \\ & \quad \mid !\, l(c).\, x(r, u).\, (\bar{r}(c).\, \bar{l}(c) \mid u(c').\, \bar{l}(c')))\end{aligned}$$

The declaration of a class will be represented as a replicator which may provide an indefinite number of private instances of the agent representing an instance of the class. In $[\![\mathsf{new}(B)]\!]$, the private name l' is a return link along which a name free in the agent representing the newly-created object may be received. In $[\![\mathsf{self}]\!]$, a is the name corresponding to the object identifier of the object containing the particular occurrence of the expression. In $[\![X]\!]$ a pair of private names is used to differentiate the operations of reading and updating the variable using the idea described above: By passing the pair (r, u) of private names and then communicating on r, $[\![X]\!]$ receives the name representing the reference stored in the variable.

The representation of a variable declaration could be simplified if recursive definitions and/or a summation operator were included in the π-calculus. We might then set:

$$[\![\mathsf{var}\ X]\!](x, c) \equiv \mathrm{Var}'(x, c)$$

$$\mathrm{Var}'(x, c) \stackrel{\mathrm{def}}{=} x(r, u).\, (\bar{r}(c).\, \mathrm{Var}'(x, c) + u(c').\, \mathrm{Var}'(x, c'))$$

The actual representation achieves a similar effect using only replication, composition and restriction. The privacy of the pair (r, u) sent by $[\![X]\!]$ and the fact that

u does not occur free in $r(c)$. Value ensure that the second component $u(c').\overline{l}(c')$ within $[\![\mathsf{var}\ X]\!]$ may be eliminated. A similar observation applies in the case of the assignment statement below with in that case the second private name u being used to update the name held. In $[\![\mathsf{var}\ X]\!] \equiv \mathrm{Var}(x, c)$, the name c will not occur as the second parameter of any agent representing an object, thus reflecting that on declaration a variable contains a reference to no object.

In translating a send expression $E_0!M(E_1, \ldots, E_n)$ and a method call $M(E_1, \ldots, E_n)$ it is necessary to capture left-to-right evaluation of parameters and to ensure that fully-evaluated subexpressions are treated correctly. We thus consider expressions $\beta_0!M(\beta_1, \ldots, \beta_n)$ and $M(\beta_1, \ldots, \beta_n)$, and $\beta_0!M(\beta_1, \ldots, \beta_{i-1}, E_i, \ldots, E_n)$ where $0 \leq i \leq n$ and $M(\beta_1, \ldots, \beta_{i-1}, E_i, \ldots, E_n)$ where $1 \leq i \leq n$ with the understanding that E_i is the first parameter whose evaluation has not yet been completed. For the send expression:

$$[\![\beta_0!M(\beta_1, \ldots, \beta_n)]\!](l, c_0, c_1, \ldots, c_n) \equiv \overline{c_0}\langle\tilde{m}\rangle.\overline{m}(c_1, \ldots, c_n, \langle l'\rangle).\mathrm{Wait}(l, l')$$

$$\begin{aligned}[\![\beta_0!M(\beta_1, \ldots, \beta_{i-1}, E_i, \ldots, E_n)]\!](l, \ldots) \equiv\ & \langle l_0, \ldots, l_i, g_{i+1}, \ldots, g_n\rangle \\ & (\mathrm{MSend}_i(l, l_0, \ldots, l_i, g_{i+1}, \ldots, g_n) \\ & \mid [\![\beta_0]\!](l_0, c_0) \\ & \mid \ldots \\ & \mid [\![\beta_{i-1}]\!](l_{i-1}, c_{i-1}) \\ & \mid [\![E_i]\!](l_i, \ldots) \\ & \mid g_{i+1}(l_{i+1}).[\![E_{i+1}]\!](l_{i+1}, \ldots) \\ & \mid \ldots \\ & \mid g_n(l_n).[\![E_n]\!](l_n, \ldots))\end{aligned}$$

where for $0 \leq i \leq n-1$:

$$\begin{aligned}\mathrm{MSend}_i \equiv\ & l_i(c_i).\ (\mathrm{Value}(l_i, c_i) \\ & \mid \overline{g_{i+1}}\langle l_{i+1}\rangle.\mathrm{MSend}_{i+1}(l, l_0, \ldots, l_{i+1}, g_{i+2}, \ldots, g_n)\end{aligned}$$

$$\begin{aligned}\mathrm{MSend}_n \equiv\ & l_n(c_n).\ (\mathrm{Value}(l_n, c_n) \\ & \mid l_0(c_0).\ \ldots.\ l_n(c_n).[\![\beta_0!M(\beta_1, \ldots, \beta_n)]\!](l, c_0, c_1, \ldots, c_n))\end{aligned}$$

Control of evaluation resides with the MSend component which activates the agents representing the subexpressions E_j in turn until they are all evaluated, then collects the names $c_0, \ldots, c_n$ representing the values, and then invokes the desired method passing the names as parameters together with a private return link for the result. To differentiate the method M, MSend sends a sequence of private names of the appropriate sorts and then communicates on the one corresponding to M. Complementary behaviour will be exhibited by the agent representing the answer statement, and as in $[\![X]\!]$ above, subcomponents guarded by unused private names may be eliminated. Having invoked the method, the sending agent waits for the result to be returned.

A local method call is somewhat similar:

$$[\![M(\beta_1,\ldots,\beta_n)]\!](l,m,c_1,\ldots,c_n) \equiv \overline{m}(c_1,\ldots,c_n,l)$$

$$\begin{aligned}[\![M(\beta_1,\ldots,\beta_{i-1},E_i,\ldots,E_n)]\!](l,m,\ldots) \equiv\ & \langle l_1,\ldots,l_i,g_{i+1},\ldots,g_n\rangle \\ & (\mathrm{MCall}_i(l,m,l_1,\ldots,l_i,g_{i+1},\ldots,g_n) \\ & \mid [\![\beta_1]\!](l_1,c_1) \\ & \mid \ldots \\ & \mid [\![\beta_{i-1}]\!](l_{i-1},c_{i-1}) \\ & \mid [\![E_i]\!](l_i,\ldots) \\ & \mid g_{i+1}(l_{i+1}).\,[\![E_{i+1}]\!](l_{i+1},\ldots) \\ & \mid \ldots \\ & \mid g_n(l_n).\,[\![E_n]\!](l_n,\ldots))\end{aligned}$$

where for $1 \leq i \leq n-1$:

$$\begin{aligned}\mathrm{MCall}_i \equiv\ & l_i(c_i).\ (\mathrm{Value}(l_i,c_i) \\ & \mid \overline{g_{i+1}}\langle l_{i+1}\rangle.\,\mathrm{MCall}_{i+1}(l,m,l_1,\ldots,l_{i+1},g_{i+2},\ldots,g_n)\end{aligned}$$

$$\begin{aligned}\mathrm{MCall}_n \equiv\ & l_n(c_n).\ (\mathrm{Value}(l_n,c_n) \\ & \mid l_1(c_1).\ldots.\,l_n(c_n).\,[\![M(\beta_1,\ldots,\beta_n)]\!](l,m,c_1,\ldots,c_n))\end{aligned}$$

MCall differs from MSend in that once the parameters have been evaluated and the names representing the values collected, in a single communication along m it: (i) requests from the replicator representing the definition of the method M in the object in which the call expression occurs (see below) a replica of the method; (ii) passes the appropriate parameter names $\tilde{c}$; and (iii) transfers the name l so that completion of the activity of the agent representing the replica of the method becomes completion of the evaluation of the call expression.

For a local declaration with $Vdecs \equiv \mathsf{var}\ \tilde{Y}$:

$$[\![\,Vdecs\ \mathsf{in}\ E]\!] \equiv \langle\tilde{y}\rangle([\![E]\!](\tilde{y},\ldots) \mid [\![\mathsf{var}\ \tilde{Y}]\!](\tilde{y},\tilde{c}))$$

where

$$[\![\mathsf{var}\ \tilde{Y}]\!] \equiv \Pi_i[\![\mathsf{var}\ Y_i]\!](y_i,c_i)$$

Thus the restriction operator captures the scope restriction of a local declaration. The final expression form $S;E$ is covered by $S_1;S_2$ below.

For the auxiliary expressions:

$$[\![\beta]\!](l,c) \equiv \mathrm{Value}(l,c)$$

$$[\![\mathsf{wait}(\beta)]\!](l,l') \equiv \mathrm{Wait}(l,l')$$

$$[\![E \Rightarrow \beta]\!](l,\ldots) \equiv \langle l''\rangle([\![E]\!](l'',\ldots) \mid \mathrm{Return}(l,l',l''))$$

$$[\![(E,\rho)]\!] \equiv \langle\tilde{y}\rangle([\![E]\!](\tilde{y}) \mid [\![\rho]\!](\tilde{y},\tilde{c}))$$

where if $\mathrm{dom}(\rho) = \tilde{Y}$:

$$[\![\rho]\!] \equiv \Pi_i \mathrm{Var}(y_i, c_i)$$

In $[\![\beta]\!]$, c will occur free in the agent representing the object β provided $\beta \neq \mathsf{nil}$; if $\beta = \mathsf{nil}$ the name c will not occur free in any such agent and will thus represent a reference to no object. As it appears in the axioms (ASSIGN1) and (RETURN2) and, in case S is not an expression, in the axiom (LOOP), nil is translated as follows:

$$[\![\mathsf{nil}]\!](l) \equiv \mathrm{Null}(l)$$

In $[\![\mathsf{wait}(\beta)]\!]$, l' is a return link which will occur free in the agent representing β, while in $[\![E \Rightarrow \beta]\!]$, on evaluation of E such a link is used to return the value to the agent representing the waiting object. $[\![(E, \rho)]\!]$ is similar to $[\![\mathit{Vdecs}\ \mathsf{in}\ E]\!]$ above.

For the simple statements:

$$[\![X := E]\!] \equiv \langle l' \rangle([\![E]\!](l', \ldots) \mid \mathrm{Assign}(l, x, l'))$$

$$\mathrm{Assign} \equiv l'(c).\, \overline{x}\langle r, u \rangle.\, \overline{u}(c).\, \mathrm{Null}(l)$$

$$[\![\mathsf{answer}]\!] \equiv a(\tilde{m}).\, \Pi_i(m_i(\tilde{c}, l').\, \overline{m_i'}(\tilde{c}, \langle l'' \rangle).\, \mathrm{Return}(l, l', l''))$$

In $[\![X := E]\!]$, when $[\![E]\!]$ completes its evaluation it communicates with Assign which then uses the private-name technique as described in $[\![X]\!]$ and $[\![\mathsf{var}\ X]\!]$ above to update the variable before signalling termination via the name l of sort NULL. Thus the effect of storing a reference in a variable is represented by passing to the agent representing the variable a name representing the reference. Again using the private-name technique, this time as described in $E_0!M(E_1, \ldots, E_n)$ above, $[\![\mathsf{answer}]\!]$ accepts a request to execute any of the available methods. As noted earlier, as the names $\tilde{m}$ are private and the sending agent will communicate on only one of them, the other components of the composition $\Pi_i\, m_i \ldots$ may be eliminated. Having received the parameters and the return link $[\![\mathsf{answer}]\!]$, like MCall above, requests from the replicator representing the appropriate method declaration an agent representing a replica of the method. $[\![\mathsf{answer}]\!]$ then evolves into Return which mediates the return of the value from the method agent to the sending agent before signalling termination via l : NULL.

For the compound statement forms we have to consider cases depending on whether or not certain substatements are expressions. For if E then S_1 else S_2 to be a well-formed statement, either S_1 and S_2 are expressions of some type C or neither is an expression:

$$\begin{aligned}[\![\mathsf{if}\ E\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2]\!](l, \ldots) \equiv \langle l', l_1, l_2 \rangle\ (&\mathrm{BoolEval}(l', l_1, l_2) \\ &\mid [\![E]\!](l', \ldots) \\ &\mid l_1.\, [\![S_1]\!](l, \ldots)\end{aligned}$$

$$\mid l_2.[\![S_2]\!](l,\ldots))$$

$$\text{BoolEval} \equiv l'(b).\overline{b}\langle\tilde{m}\rangle.\overline{m_1}\langle t,f\rangle.(t.\overline{l_1} \mid f.\overline{l_2})$$

where l is of sort LINK[C] for some type C or, like l_1 and l_2, it is of sort NULL.

For S_1 an expression of type C:

$$[\![S_1;S_2]\!](l,\ldots) \equiv \langle l'\rangle([\![S_1]\!](l',\ldots) \mid l'(c).[\![S_2]\!](l,\ldots))$$

where l' : LINK[C] and $c \notin \text{fn}([\![S_2]\!])$.

For S_1 a statement which is not an expression:

$$[\![S_1;S_2]\!](l,\ldots) \equiv \langle l'\rangle([\![S_1]\!](l',\ldots) \mid l'.[\![S_2]\!](l,\ldots))$$

where l' : NULL.

For S an expression of type C:

$$[\![\textsf{while } E \textsf{ do } S]\!](l,\ldots) \equiv \langle l'\rangle(W(l,l',\ldots) \mid !\,l'(c').W(l,l',\ldots))$$

$$\begin{aligned} W \equiv \langle l'',l_1,l_2\rangle\ (&\text{BoolEval}(l'',l_1,l_2)\\ &\mid [\![E]\!](l'',\ldots)\\ &\mid l_1.[\![S]\!](l',\ldots)\\ &\mid l_2.\text{Value}(l,c)) \end{aligned}$$

where l' : LINK[C], $c' \notin \text{fn}(W)$ and c represents a reference to no object.

For S a statement which is not an expression:

$$[\![\textsf{while } E \textsf{ do } S]\!](l,\ldots) \equiv \langle l'\rangle(W(l,l',\ldots) \mid !\,l'.W(l,l',\ldots))$$

$$\begin{aligned} W \equiv \langle l'',l_1,l_2\rangle\ (&\text{BoolEval}(l'',l_1,l_2)\\ &\mid [\![E]\!](l'',\ldots)\\ &\mid l_1.[\![S]\!](l',\ldots)\\ &\mid l_2.\text{Null}(l)) \end{aligned}$$

where l' : NULL.

In $[\![\textsf{if } E \textsf{ then } S_1 \textsf{ else } S_2]\!]$, on evaluation of the expression, BoolEval receives from $[\![E]\!]$ a link to the agent representing the Boolean value, determines which it is (see the definitions of the agents representing the standard objects below), and activates the appropriate component; the other component may be eliminated as usual. In $[\![S_1;S_2]\!]$ the agent $[\![S_2]\!]$ begins to act on termination of the agent representing S_1; the value of S_1, if it is an expression, is ignored by $[\![S_2]\!]$. In $[\![\textsf{while } E \textsf{ do } S]\!]$, repeated evaluation of E and execution of S is mimicked using a replication in a style similar to that used in the definition of $[\![\textsf{var } X]\!]$ above.

It remains to consider method, class and unit declarations. Suppose

$$Mdec \equiv \text{method } M(\tilde{Y} : \tilde{C}) : C \text{ is } E$$

$$Mdecs \equiv Mdec_1, \ldots, Mdec_q$$

$$Cdec \equiv \text{class } A \text{ is } Vdecs, Mdecs \text{ with body } S$$

$$Udec \equiv \text{unit } U \text{ is } Cdec_1, \ldots, Cdec_r \text{ with root } A$$

Then

$$[\![Mdec]\!] \equiv \,!\,\mathrm{MMethod}(m, a, \tilde{x}, \tilde{m}, \tilde{k}, \tilde{c})$$

$$\begin{aligned}\mathrm{MMethod} \equiv\ & \langle g, \tilde{y}\rangle\ (\mathrm{MHandle}(m, g, \tilde{y}) \\ & \mid [\![\text{var } \tilde{Y}]\!](\tilde{y}, \tilde{c}) \\ & \mid g(l).\,[\![E]\!](l, \ldots))\end{aligned}$$

$$\mathrm{MHandle} \equiv m(\tilde{c}, l).\,\overline{y_1}\langle r_1, u_1\rangle.\,\overline{u_1}(c_1).\,\ldots.\,\overline{y_n}\langle r_n, u_n\rangle.\,\overline{u_n}(c_n).\,\overline{g}(l)$$

$$[\![Mdecs]\!] \equiv \Pi_i [\![Mdec_i]\!]$$

Thus a method declaration is represented as a replicator which may generate agents each representing a replica of the method. The component MHandle receives the names representing the parameters and the name to be used on evaluation of the method body. The second component represents a local store in which MHandle stores the parameter names before activating the expression agent.

For a class declaration:

$$[\![Cdec]\!](k, \tilde{k}, \tilde{c}) \equiv\ !\,k(l).\,\bar{l}\langle a\rangle.\,\mathrm{NewObject}(a, \tilde{k}, \tilde{c})$$

$$\begin{aligned}\mathrm{NewObject} \equiv\ & \langle l, \tilde{x}, \tilde{m}\rangle\ ([\![S]\!](l, a, \tilde{x}, \tilde{m}, \tilde{k}, \tilde{c}) \\ & \mid [\![Vdecs]\!](\tilde{x}, \tilde{c}) \\ & \mid [\![Mdecs]\!](\tilde{m}, a, \tilde{x}, \tilde{k}, \tilde{c}))\end{aligned}$$

Thus a class declaration is represented as a replicator which may generate agents each representing an instance of the class. Each such agent has a name which plays the rôle of an object identifier. This name is initially private to NewObject and the agent representing the object containing the expression new(A) whose evaluation resulted in the activation of the replicator.

Finally, for a unit declaration:

$$[\![Udec]\!] \equiv \langle \tilde{k}\rangle(\mathrm{Classes}(\tilde{k}) \mid \langle a, \tilde{c}\rangle \mathrm{NewObject}(a, \tilde{k}, \tilde{c}))$$

$$\mathrm{Classes} \equiv \Pi_i [\![Cdec_i]\!] \mid \mathrm{Boolean}$$

where Boolean representing the standard class Bool and the standard objects tt and ff is defined below.

A unit declaration is represented as a composition of the replicators representing the classes together with an agent representing the root object.

6.1 The Standard Class

The representations of the standard class Bool and of the standard objects tt and ff are the components BoolClass, Bool_t and Bool_f of the agent Boolean defined as follows:

$$\begin{aligned}
\text{Boolean}(bk) \;\equiv\; & \langle b_t, b_f\rangle\ (\text{BoolClass}(bk, b_t, b_f) \\
& \mid \text{Bool}_t(b_t, bk) \\
& \mid \text{Bool}_f(b_f, bk))
\end{aligned}$$

$$\text{BoolClass}(bk, b_t, b_f) \;\equiv\; !\,bk(l,t,f).\,(t.\,\overline{l}(b_t) \mid f.\,\overline{l}(b_f))$$

$$\begin{aligned}
\text{Bool}_v(b, bk) \;\equiv\; & \langle \tilde{m}\rangle\ (!\,\text{BoolBody}(b, \tilde{m}) \\
& \mid !\,\text{BoolVal}_v(m_1) \\
& \mid !\,\text{BoolNot}(m_2, m_1, bk) \\
& \mid !\,\text{BoolAnd}(m_3, m_1, bk))
\end{aligned}$$

$$\begin{aligned}
\text{BoolBody}(b, \tilde{m}) \;\equiv\; & b(\tilde{m'}).\ (m_1'(t,f).\,\overline{m_1}(t,f) \\
& \mid m_2'(l).\,\overline{m_2}(l) \\
& \mid m_3'(b',l).\,\overline{m_3}(b',l))
\end{aligned}$$

$$\text{BoolVal}_t(m_1) \;\equiv\; m_1(t,f).\,\overline{t}$$

$$\text{BoolVal}_f(m_1) \;\equiv\; m_1(t,f).\,\overline{f}$$

$$\begin{aligned}
\text{BoolNot}(m_2, m_1, bk) \;\equiv\; & m_2(l).\,\overline{m_1}\langle t,f\rangle.\ (t.\,\overline{bk}(l,\langle t,f\rangle).\,\overline{f} \\
& \mid f.\,\overline{bk}(l,\langle t,f\rangle).\,\overline{t})
\end{aligned}$$

$$\begin{aligned}
\text{BoolAnd}(m_3, m_1, bk) \;\equiv\; & m_3(b',l).\,\overline{m_1}\langle t,f\rangle. \\
& (f.\,\overline{bk}(l,\langle t,f\rangle).\,\overline{f} \\
& \mid t.\,b'(\tilde{m'}).\,\overline{m_1'}\langle t',f'\rangle.\ (t'.\,\overline{bk}(l,\langle t,f\rangle).\,\overline{t} \\
& \qquad \mid f'.\,\overline{bk}(l,\langle t,f\rangle).\,\overline{f}))
\end{aligned}$$

These representations employ ideas seen earlier in the translation. Note in particular how [[true]] and [[false]] interact with BoolClass and BoolEval interacts with Bool_t and Bool_f.

7 The Correspondence

The translation can be extended to a mapping from configurations to agents. Fix a unit U. Let $\langle \Phi^0, \Gamma^0\rangle$ be the initial configuration associated with U, α_0 the name of the root object, and A_0 the name of the root class. Suppose that

$$\langle \Phi^0, \Gamma^0\rangle \longrightarrow \langle \Phi^1, \Gamma^1\rangle \longrightarrow \ldots \longrightarrow \langle \Phi^n, \Gamma^n\rangle = \langle \Phi, \Gamma\rangle$$

Lemma 1 Let $\langle i_1, \ldots, i_p \rangle$ be the list of those i such that the transition $\langle \Phi^i, \Gamma^i \rangle \longrightarrow \langle \Phi^{i+1}, \Gamma^{i+1} \rangle$ is inferred using the rule (NEW). Let $\langle (A_i, \alpha_i) \mid 1 \leq i \leq p \rangle$ be the corresponding list of labels on the arrows in the premises of the instances of the rules. Then the α_i $(1 \leq i \leq p)$ are all distinct from one another and from α_0, and $\Phi_2 = \{(\alpha_i, A_i) \mid 0 \leq i \leq p\}$. Moreover, setting $N = \mathrm{dom}(\Phi_2)$, $\Gamma = \{\langle \alpha, S_\alpha, \sigma_\alpha \rangle \mid \alpha \in N\}$ for some S_α and σ_α.

Proof: A straightforward induction on n using properties of the transition rules. The details are omitted. □

Lemma 2 In the configuration $\langle \Phi, \Gamma \rangle$ let

$$W = \{\alpha \in N \mid S_\alpha \text{ has a substatement of the form } \mathsf{wait}(\gamma)\}$$
$$R = \{\beta \in N \mid S_\beta \text{ has a substatement of the form } E \Rightarrow \gamma\}$$
$$I = \{\langle \alpha, \beta \rangle \in W \times R \mid S_\alpha \text{ has } \mathsf{wait}(\beta) \text{ as a substatement, and } S_\beta \text{ has a substatement of the form } E \Rightarrow \alpha\}$$

Then I is a total onto function from W to R. Furthermore if $\langle \alpha, \beta \rangle \in I$ then:

1. $\mathsf{wait}(\beta)$ occurs exactly once as a substatement of S_α, and S_β has exactly one substatement of the form $E \Rightarrow \alpha$.
2. $\langle \alpha, \mu_\alpha \rangle \vdash \langle S_\alpha, \sigma_\alpha \rangle \stackrel{\ell}{\longmapsto} \langle S'_\alpha, \sigma'_\alpha \rangle$ iff $\ell = \mathrm{result}(\beta, \gamma)$ for some γ, S'_α contains no substatement of the form $\mathsf{wait}(\gamma)$, and $\sigma'_\alpha = \sigma_\alpha$.
3. If $\langle \beta, \mu_\beta \rangle \vdash \langle S_\beta, \sigma_\beta \rangle \stackrel{\ell}{\longmapsto} \langle S'_\beta, \sigma'_\beta \rangle$ then either (i) $\ell = \mathrm{return}(\alpha, \gamma)$ for some γ, S'_β contains no substatement of the form $E \Rightarrow \alpha$, and $\sigma'_\beta = \sigma_\beta$, or (ii) ℓ is of another form and S'_β does contain a substatement of the form $E \Rightarrow \alpha$.

Proof: A straightforward induction on n using properties of the transition rules. The details are omitted. □

The following lemmas characterize the sorts of the names occurring free in translations of syntactic entities. Their proofs are straightforward.

Lemma 3 Suppose $Vdecs \equiv \mathsf{var}\ Y_1 : C_1, \ldots, Y_n : C_n$. The names occurring free in $[\![Vdecs]\!]$ are:

1. $\tilde{y} : \mathsf{VARS} = \mathsf{VAR}[C_1], \ldots, \mathsf{VAR}[C_n]$, and
2. $\tilde{c} : \mathsf{OBJECTS} = \mathsf{OBJECT}[C_1], \ldots, \mathsf{OBJECT}[C_n]$.

Proof: Directly from the definition. □

Fix $\langle \alpha, S, \sigma \rangle \in \Gamma$. Suppose $\Phi_2(\alpha) = A$ and $\Phi_1(A) = \langle \tilde{X}, \mu, S^0 \rangle$.

Lemma 4 The names occurring free in $[\![S]\!]$ are $(l, a, \tilde{x}, \tilde{m}, \tilde{k}, \tilde{c}, \tilde{l})$ with sorts as follows:

1. If S is an expression of type C then l : LINK[C]; if S is not an expression then l : NULL,
2. a : OBJECT[A], S being the yet-to-be-executed statement of an object of class A,
3. $\tilde{x}$: VARS = VAR[C_1], ..., VAR[C_n] where $C_1, \ldots, C_n \in \mathrm{dom}(\Phi_1)$,
4. $\tilde{m}$: METHODS[A], S being the yet-to-be-executed statement of an object of class A,
5. $\tilde{k}$: CLASSES = CLASS[C_1], ..., CLASS[C_n] where $C_1, \ldots, C_n \in \mathrm{dom}(\Phi_1)$,
6. $\tilde{c}$: OBJECTS = OBJECT[C_1], ..., OBJECT[C_n] where $C_1, \ldots, C_n \in \mathrm{dom}(\Phi_1)$, and
7. $\tilde{l}$: LINKS = LINK[C_1], ..., LINK[C_n] where $C_1, \ldots, C_n \in \mathrm{dom}(\Phi_1)$.

Proof: By induction on S. The details are straightforward given the definitions and are omitted. □

Lemma 5 The names occurring free in $[\![\sigma]\!]$ are $(\tilde{y}, \tilde{c})$ with sorts:

1. $\tilde{y}$: VARS = VAR[C_1], ..., VAR[C_n], and
2. $\tilde{c}$: OBJECTS = OBJECT[C_1], ..., OBJECT[C_n]

where $\mathrm{dom}(\sigma) = \{Y_1 : C_1, \ldots, Y_n : C_n\}$.

Proof: Directly from the definitions. □

Lemma 6 The names occurring free in $[\![\mu]\!]$ are $(\tilde{m}, a, \tilde{x}, \tilde{k}, \tilde{c})$ with sorts:

1. $\tilde{m}$: METHODS[A],
2. a : OBJECT[A],
3. $\tilde{x}$: VARS = VAR[C_1], ..., VAR[C_n] where $C_1, \ldots, C_n \in \mathrm{dom}(\Phi_1)$,
4. $\tilde{k}$: CLASSES = CLASS[C_1], ..., CLASS[C_n] where $C_1, \ldots, C_n \in \mathrm{dom}(\Phi_1)$, and
5. $\tilde{c}$: OBJECTS = OBJECT[C_1], ..., OBJECT[C_n] where $C_1, \ldots, C_n \in \mathrm{dom}(\Phi_1)$.

Proof: By definition, $[\![\mu]\!] = [\![Mdecs]\!]$ where *Mdecs* is the appropriate sequence of method declarations. The result follows from the definitions and the previous lemmas. □

Lemma 7 The agent Classes representing the class declarations of U has free names $\tilde{k}$: CLASSES = CLASS[C_1], ..., CLASS[C_n] where $\{C_1, \ldots, C_n\} = \mathrm{dom}(\Phi_1)$.

Proof: The result follows from the definitions and the previous lemmas. □

The object α is represented as the agent $\mathrm{Obj}_\alpha \equiv [\![\langle \alpha, S, \sigma, \mu\rangle]\!]$ defined as follows:

$$\begin{aligned}\mathrm{Obj}_\alpha(a, \tilde{k}, \tilde{c}, \tilde{l}) \equiv \langle l, \tilde{x}, \tilde{m}\rangle\ & ([\![S]\!](l, a, \tilde{x}, \tilde{m}, \tilde{k}, \tilde{c}, \tilde{l}) \\ & \mid [\![\sigma]\!](\tilde{x}, \tilde{c}) \\ & \mid [\![\mu]\!](\tilde{m}, a, \tilde{x}, \tilde{k}, \tilde{c}))\end{aligned}$$

To define the mapping from configurations to agents let $\Gamma = \{\langle \alpha, S_\alpha, \sigma_\alpha\rangle \mid \alpha \in N\}$ where for $\alpha \in N = \mathrm{dom}(\Phi_2)$, $\Phi_2(\alpha) = A_\alpha$ and $\Phi_1(A_\alpha) = \langle \tilde{X}_\alpha, \mu_\alpha, S^0_\alpha\rangle$. Choose names so that setting

$$\mathrm{Objects}(\tilde{k}) \equiv \langle \tilde{a}, \tilde{c}, \tilde{l}\rangle\, \Pi_{\alpha\in N}\, \mathrm{Obj}_\alpha(a, \tilde{k}, \tilde{c}, \tilde{l})$$

the sharing of names by agents mirrors the disposition of references and the associations among objects due to incomplete method invocations described in Lemma 2.

Finally define

$$[\![\langle \Phi, \Gamma\rangle]\!] \equiv \langle \tilde{k}\rangle(\mathrm{Objects}(\tilde{k}) \mid \mathrm{Classes}(\tilde{k}))$$

We now state and prove a correspondence between the semantics.

Theorem 1 Suppose U is a unit and $\langle \Phi^0, \Gamma^0\rangle$ is the initial configuration associated with U.

1. Any computation $\langle \Phi^0, \Gamma^0\rangle \longrightarrow \langle \Phi^1, \Gamma^1\rangle \longrightarrow \ldots \longrightarrow \langle \Phi^n, \Gamma^n\rangle$ is directly mirrored by a derivation $[\![\langle \Phi^0, \Gamma^0\rangle]\!] \Longrightarrow\sim [\![\langle \Phi^1, \Gamma^1\rangle]\!] \Longrightarrow\sim \ldots \Longrightarrow\sim [\![\langle \Phi^n, \Gamma^n\rangle]\!]$.
2. If $[\![\langle \Phi^0, \Gamma^0\rangle]\!] \Longrightarrow P$ then for some $\langle \Phi, \Gamma\rangle$, $\langle \Phi^0, \Gamma^0\rangle \longrightarrow^* \langle \Phi, \Gamma\rangle$ and $P \Longrightarrow\sim [\![\langle \Phi, \Gamma\rangle]\!]$.

Proof: The proof is lengthy. We begin the proof of the first part with three preliminary lemmas.

Lemma 8 Let S be a statement and $[\![S]\!] \equiv [\![S]\!](l, \ldots)$. Suppose $[\![S]\!] \xrightarrow{w}\xrightarrow{\alpha} P$ where l occurs in the action α but not in the sequence of actions w. Then $P \equiv \mathbf{0}$ and:

1. If S is not an expression then $\alpha = \overline{l}$.
2. If S is an expression then (i) $\alpha = \overline{l}(c)$ or $\overline{l}\langle c\rangle$, or (ii) $\alpha = \overline{m}(\tilde{c}, l)$.

Proof: By induction on the structure of S.

In the cases $S \equiv \mathsf{true}, \mathsf{false}, \mathsf{new}(B), \mathsf{self}, X, \beta, \mathsf{wait}(\beta)$, it follows directly from the definitions that $P \equiv \mathbf{0}$ and 2(i) holds.

If $S \equiv E!M(\tilde{E})$ then from the definitions it follows that 2(i) holds with α being the last action of MSend and also that $P \equiv \mathbf{0}$.

If $S \equiv M(\tilde{E})$ then from the definitions it follows that 2(ii) holds with α being the last action of MCall and also that $P \equiv \mathbf{0}$.

If $S \equiv \mathit{Vdecs}$ in E or (E, ρ) then by induction hypothesis α is of one of the stated forms and $P \equiv \langle \tilde{y} \rangle (\mathbf{0} \mid [\![\rho]\!](\tilde{y}, \tilde{c})) \equiv \mathbf{0}$.

If $S \equiv E \Rightarrow \beta$ or $X := E$ then from the definitions it follows that 1 holds with α being the last action of Return or Assign respectively and by induction hypothesis $P \equiv \mathbf{0}$.

If $S \equiv$ answer then from the definitions it follows that 1 holds with α being the last action of Return and also that $P \equiv \mathbf{0}$.

If $S \equiv$ if E then S_1 else S_2 then from the definitions and the induction hypothesis it follows that α is of one of the stated forms and that either $P \equiv \langle l', l_1, l_2, f \rangle (f.\overline{l_2} \mid \mathbf{0} \mid \mathbf{0} \mid l_2.[\![S_2]\!]) \equiv \mathbf{0}$ or $P \equiv \langle l', l_1, l_2, t \rangle (t.\overline{l_1} \mid \mathbf{0} \mid l_1.[\![S_1]\!] \mid \mathbf{0}) \equiv \mathbf{0}$.

If $S \equiv S_1; S_2$ then again by induction hypothesis α is of one of the stated forms and $P \equiv \langle l' \rangle (\mathbf{0} \mid \mathbf{0}) \equiv \mathbf{0}$.

If $S \equiv$ while E do S' then from the definitions and the induction hypothesis it follows that α is of one of the stated forms and that for some $n \geq 1$,

$$P \equiv \langle l', l'', l_1, l_2, f \rangle ((f.\overline{l_2} \mid \mathbf{0} \mid \mathbf{0} \mid l_2.\mathrm{Null})^n \mid !\, l'.W(l, l', \ldots)) \equiv \mathbf{0}$$

This completes the proof. □

Lemma 9 Let S be a statement and $[\![S]\!] \equiv [\![S]\!](l, \ldots)$. Suppose μ is a family of method definitions appropriate for S and that $\langle \tilde{m} \rangle ([\![S]\!] \mid [\![\mu]\!]) \xrightarrow{w} \xrightarrow{\alpha} P$ where l occurs in α but not in w. Then $P \equiv \mathbf{0}$ and:

1. If S is not an expression then $\alpha = \overline{l}$.
2. If S is an expression then $\alpha = \overline{l}(c)$ or $\overline{l}\langle c \rangle$.

Proof: From the definitions and the preceding lemma it follows that α must be of the stated form and that $P \equiv \langle \tilde{m} \rangle (\mathbf{0} \mid [\![\mu]\!]) \equiv \mathbf{0}$. □

Lemma 10 For μ an appropriate family of method definitions,

$$\langle \tilde{m} \rangle ([\![\text{while } E \text{ do } S]\!] \mid [\![\mu]\!]) \sim \langle \tilde{m} \rangle ([\![\text{if } E \text{ then } (S; \text{while } E \text{ do } S) \text{ else nil}]\!] \mid [\![\mu]\!])$$

Proof: Assuming S is not an expression (the other case is similar) we have

$$[\![\text{while } E \text{ do } S]\!](l,\ldots) \equiv \langle l'\rangle(W(l,l',\ldots) \mid !\, l'.\, W(l,l',\ldots))$$
$$\equiv \langle l'\rangle(W(l,l',\ldots) \mid l'.\, W(l,l',\ldots) \mid !\, l'.\, W(l,l',\ldots))$$

and

$$[\![\text{if } E \text{ then } (S; \text{while } E \text{ do } S) \text{ else nil}]\!](l,\ldots) \equiv \langle l', l''\rangle\ (W(l,l'',\ldots) \mid l''.\, W(l,l',\ldots) \mid !\, l'.\, W(l,l',\ldots))$$

where

$$W(l,l',\ldots) \equiv \langle l_0,l_1,l_2\rangle\ (\mathrm{BoolEval}(l_0,l_1,l_2) \mid [\![E]\!](l_0,\ldots) \mid l_1.\, [\![S]\!](l',\ldots) \mid l_2.\, \mathrm{Null}(l))$$

The only difference between the two agents is thus the appearance of l'' in the latter. But since this name is the first parameter of $[\![S]\!]$ in the first component W, from the preceding lemma it follows that the first instance of W may activate only the second component and then evolve into **0**. □

To prove the first part of the theorem it suffices to establish the following:

Claim Suppose $\langle \Phi^0, \Gamma^0\rangle \longrightarrow^* \langle \Phi, \Gamma\rangle \longrightarrow \langle \Phi', \Gamma'\rangle$. Then $[\![\langle \Phi, \Gamma\rangle]\!] \Longrightarrow\sim [\![\langle \Phi', \Gamma'\rangle]\!]$.

To prove the claim we consider four cases, one for each of the global transition rules which may be used to infer the transition $\langle \Phi, \Gamma\rangle \longrightarrow \langle \Phi', \Gamma'\rangle$.

Lemma 11 (LOCAL) Suppose the transition is inferred by

$$\frac{\langle \alpha, \mu\rangle \vdash \langle S, \sigma\rangle \longmapsto \langle S', \sigma'\rangle}{\langle \Phi, \Gamma \cup \{\langle \alpha, S, \sigma\rangle\}\rangle \longrightarrow \langle \Phi, \Gamma \cup \{\langle \alpha, S', \sigma'\rangle\}\rangle}$$

Then

$$\mathrm{Obj}_\alpha \mid \mathrm{Boolean} \Longrightarrow\sim \mathrm{Obj}'_\alpha \mid \mathrm{Boolean}$$

where $\mathrm{Obj}_\alpha \equiv [\![\langle \alpha, S, \sigma, \mu\rangle]\!]$ and $\mathrm{Obj}'_\alpha \equiv [\![\langle \alpha, S', \sigma', \mu\rangle]\!]$.

Proof: By induction on the depth of inference of the premise in the local transition system. We consider in turn the axioms and rules which could be applied in the last step of this inference. (NEW), (SEND1), (ANSWER), (WAIT) and (RETURN2) are not applicable.

(SELF) We have $S \equiv \mathsf{self}$, $S' \equiv \alpha$ and $\sigma' = \sigma$. Since $[\![\mathsf{self}]\!] \equiv [\![\alpha]\!]$ the result follows directly.

(VAR) We have $S \equiv X$, $S' \equiv \beta$ where $\beta = \sigma(X)$, and $\sigma' = \sigma$. Setting $v = \overline{x}\langle r, u\rangle.\, r(c)$ and $w = x(r, u).\, \overline{r}(c)$ we have $[\![X]\!] \stackrel{v}{\Longrightarrow} [\![\beta]\!]$ and $[\![\sigma]\!] \stackrel{w}{\Longrightarrow} [\![\sigma]\!]$ and the result follows.

(CALL1) We have $S \equiv M(\tilde{\beta})$, $S' \equiv (E, \rho)$ where $\mu(M) = \langle \tilde{Y}, E\rangle$ and $\rho = \{\tilde{Y} := \tilde{\beta}\}$, and $\sigma' = \sigma$. Since $[\![M(\tilde{\beta})]\!] \mid [\![\mu]\!] \Longrightarrow \langle \tilde{y}\rangle([\![E]\!](\tilde{y}) \mid [\![\rho]\!](\tilde{y}, \tilde{c})) \mid [\![\mu]\!]$, the result follows.

(VARDEC) We have $S \equiv \mathit{Vdecs}$ in E, $S' \equiv (E, \rho)$ where $\rho = \{\tilde{Y} := \mathsf{nil}\}$ where $\mathit{Vdecs} \equiv \mathsf{var}\ \tilde{Y}$, and $\sigma' = \sigma$. Since $[\![\mathit{Vdecs}\ \mathsf{in}\ E]\!] \equiv [\![(E, \rho)]\!]$, the result follows directly.

(ASSIGN1) We have $S \equiv X := \beta$, $S' \equiv \mathsf{nil}$ and $\sigma' = \sigma[X := \beta]$. Since $[\![X := \beta]\!] \mid [\![\sigma]\!] \Longrightarrow [\![\mathsf{nil}]\!] \mid [\![\sigma']\!]$, the result follows.

(SEQ1) We have $S \equiv \beta; S'$ and $\sigma' = \sigma$. The result follows directly from the definition since $[\![\beta; S']\!] \Longrightarrow [\![S']\!]$.

(COND1) We have $S \equiv \mathsf{if\ tt\ then}\ S'\ \mathsf{else}\ S_2$ and $\sigma' = \sigma$. We have

$$[\![S]\!] \mid \mathrm{Boolean} \Longrightarrow \langle l_1, l_2, f\rangle(f.\,\overline{l_2} \mid [\![S']\!] \mid l_2.\,[\![S_2]\!]) \mid \mathrm{Boolean} \equiv [\![S']\!] \mid \mathrm{Boolean}$$

and the result follows.

(COND2) Similar to (COND1).

(LOOP) We have $S \equiv \mathsf{while}\ E\ \mathsf{do}\ S''$, $S' \equiv \mathsf{if}\ E\ \mathsf{then}\ (S''; S)\ \mathsf{else\ nil}$ and $\sigma' = \sigma$. The result follows directly from Lemma 10.

(NOT) We have $S \equiv \mathsf{b!\ not}\ ()$, $S' \equiv \mathsf{b}'$ where $b' = \neg b$, and $\sigma' = \sigma$. By examining the definitions in detail we see that if $\mathsf{b} \equiv \mathsf{tt}$ then

$$\begin{array}{ll} & [\![\mathsf{b!\ not}\ ()]\!] \mid \mathrm{Boolean} \\ \Longrightarrow & \langle l'\rangle(\mathrm{Wait}(l, l') \mid \langle t, f\rangle(\overline{bk}(l', \langle t', f'\rangle).\,\overline{f'} \mid f.\,\overline{bk}(l', \langle t', f'\rangle).\,\overline{t'})) \mid \mathrm{Boolean} \\ \sim & [\![\mathsf{b}']\!] \mid \mathrm{Boolean} \end{array}$$

and the result follows. The case $\mathsf{b} \equiv \mathsf{ff}$ is similar.

(AND) Similar to (NOT).

(RETURN1) We have $S \equiv (\gamma, \rho)$, $S' \equiv \gamma$ and $\sigma' = \sigma$. Since $[\![(\gamma, \rho)]\!] \equiv \langle \tilde{y}\rangle(\mathrm{Value}(l, c) \mid [\![\rho]\!](\tilde{y}, \tilde{c})) \equiv [\![\gamma]\!]$, the result follows directly.

(SEND2) We have $S \equiv E!M(\tilde{E})$ and $S' \equiv E'!M(\tilde{E})$ where $\langle \alpha, \mu\rangle \vdash \langle E, \sigma\rangle \longmapsto \langle E', \sigma'\rangle$ by a shorter inference. By induction hypothesis

$$[\![\langle \alpha, E, \sigma, \mu\rangle]\!] \mid \mathrm{Boolean} \Longrightarrow\sim [\![\langle \alpha, E', \sigma', \mu\rangle]\!] \mid \mathrm{Boolean}$$

and from the definitions, noting in particular the case $E' \equiv \beta$, it follows that

$$[\![\langle \alpha, S, \sigma, \mu\rangle]\!] \mid \mathrm{Boolean} \Longrightarrow\sim [\![\langle \alpha, S', \sigma', \mu\rangle]\!] \mid \mathrm{Boolean}$$

The arguments for the rules (SEND3), (CALL2), (RETURN3), (ASSIGN2), (COND3) and (SEQ2) are similar in style to that for (SEND2).

(LOCAL) In this case the result follows directly since $[\![\langle \alpha, (E, \rho), \sigma, \mu \rangle]\!] \equiv [\![\langle \alpha, E, \sigma \oplus \rho, \mu \rangle]\!]$ and $[\![\langle \alpha, (E', \rho'), \sigma' \cup \sigma'', \mu \rangle]\!] \equiv [\![\langle \alpha, E', \tau, \mu \rangle]\!]$.

This completes the proof of the lemma. □

Lemma 12 (NEW) Suppose the transition is inferred by

$$\frac{\langle \alpha, \mu \rangle \vdash \langle S, \sigma \rangle \overset{\mathsf{new}(B,\beta)}{\longmapsto} \langle S', \sigma' \rangle}{\langle \Phi, \Gamma \cup \{\langle \alpha, S, \sigma \rangle\} \rangle \longrightarrow \langle \Phi', \Gamma \cup \{\langle \alpha, S', \sigma' \rangle, \langle \beta, S'', \sigma'' \rangle\} \rangle}$$

Then

$$\mathrm{Obj}_\alpha \mid \mathrm{Classes} \Longrightarrow\sim \mathrm{Obj}'_\alpha \mid \mathrm{NewObject}_\beta \mid \mathrm{Classes}$$

where $\mathrm{Obj}_\alpha \equiv [\![\langle \alpha, S, \sigma, \mu \rangle]\!]$, $\mathrm{Obj}'_\alpha \equiv [\![\langle \alpha, S', \sigma', \mu \rangle]\!]$, $\mathrm{NewObject}_\beta \equiv [\![\langle \beta, S'', \sigma'', \mu_\beta \rangle]\!]$.

Proof: By induction on the depth of inference of the premise.

(NEW) We have $S \equiv \mathsf{new}(B)$, $S' \equiv \beta$ and $\sigma' = \sigma$. Since

$$[\![S]\!] \mid \mathrm{Classes} \Longrightarrow\sim \langle c \rangle(\mathrm{Value}(l, c) \mid \mathrm{NewObject}_\beta(c, \ldots)) \mid \mathrm{Classes}$$

the result follows.

The only other cases are the rules (SEND2)–(LOCAL). In these cases the result follows by an inductive argument as in the proof of the case (SEND2) in the preceding lemma. □

Lemma 13 (COM1) Suppose the transition is inferred by

$$\frac{\langle \alpha, \mu_\alpha \rangle \vdash \langle S_\alpha, \sigma_\alpha \rangle \overset{\mathsf{send}(\beta,M,\tilde{\beta})}{\longmapsto} \langle S'_\alpha, \sigma'_\alpha \rangle \quad \langle \beta, \mu_\beta \rangle \vdash \langle S_\beta, \sigma_\beta \rangle \overset{\mathsf{answer}(\alpha,M,\tilde{\beta})}{\longmapsto} \langle S'_\beta, \sigma'_\beta \rangle}{\langle \Phi, \Gamma \cup \{\langle \alpha, S_\alpha, \sigma_\alpha \rangle, \langle \beta, S_\beta, \sigma_\beta \rangle\} \rangle \longrightarrow \langle \Phi, \Gamma \cup \{\langle \alpha, S'_\alpha, \sigma'_\alpha \rangle, \langle \beta, S'_\beta, \sigma'_\beta \rangle\} \rangle}$$

Then

$$\mathrm{Obj}_\alpha \mid \mathrm{Obj}_\beta \Longrightarrow\sim \mathrm{Obj}'_\alpha \mid \mathrm{Obj}'_\beta$$

where $\mathrm{Obj}_\alpha \equiv [\![\langle \alpha, S_\alpha, \sigma_\alpha, \mu_\alpha \rangle]\!]$, $\mathrm{Obj}'_\alpha \equiv [\![\langle \alpha, S'_\alpha, \sigma'_\alpha, \mu_\alpha \rangle]\!]$, and similarly for β.

Proof: By simultaneous induction on the depth of inference of the premises.

Suppose the premises are inferred by (SEND1) and (ANSWER) respectively so that $S_\alpha \equiv \beta!M(\tilde{\beta})$, $S'_\alpha \equiv \mathsf{wait}(\beta)$ and $\sigma'_\alpha \equiv \sigma_\alpha$, and $S_\beta \equiv \mathsf{answer}$, $S'_\beta \equiv (E, \rho) \Rightarrow \alpha$ where $\mu_\beta(M) = \langle \tilde{Y}, E \rangle$ and $\rho = \{\tilde{Y} := \tilde{\beta}\}$, and $\sigma'_\beta = \sigma_\beta$. Setting $v = \overline{c}\langle \tilde{m} \rangle.\, \overline{m}(\tilde{c}, \langle l' \rangle)$ and $w = c(\tilde{m}).\, m(\tilde{c}, l')$, $[\![\beta!M(\tilde{\beta})]\!] \overset{v}{\Longrightarrow} [\![\mathsf{wait}(\beta)]\!]$ and

$$\mathsf{answer} \mid [\![\mu_\beta]\!] \stackrel{w}{\Longrightarrow} \mathrm{Return} \mid [\![(E,\rho)]\!] \mid [\![\mu_\beta]\!] \equiv [\![(E,\rho) \Rightarrow \alpha]\!] \mid [\![\mu_\beta]\!]$$

and so the result follows.

The remaining cases follow by similar inductive arguments to those seen earlier. □

Lemma 14 (COM2) Suppose the transition is inferred by

$$\frac{\langle \alpha, \mu_\alpha \rangle \vdash \langle S_\alpha, \sigma_\alpha \rangle \stackrel{\mathrm{result}(\beta,\gamma)}{\longmapsto} \langle S'_\alpha, \sigma'_\alpha \rangle \qquad \langle \beta, \mu_\beta \rangle \vdash \langle S_\beta, \sigma_\beta \rangle \stackrel{\mathrm{return}(\alpha,\gamma)}{\longmapsto} \langle S'_\beta, \sigma'_\beta \rangle}{\langle \Phi, \Gamma \cup \{\langle \alpha, S_\alpha, \sigma_\alpha \rangle, \langle \beta, S_\beta, \sigma_\beta \rangle\} \rangle \longrightarrow \langle \Phi, \Gamma \cup \{\langle \alpha, S'_\alpha, \sigma'_\alpha \rangle, \langle \beta, S'_\beta, \sigma'_\beta \rangle\} \rangle}$$

Then

$$\mathrm{Obj}_\alpha \mid \mathrm{Obj}_\beta \Longrightarrow\sim \mathrm{Obj}'_\alpha \mid \mathrm{Obj}'_\beta$$

where $\mathrm{Obj}_\alpha \equiv [\![\langle \alpha, S_\alpha, \sigma_\alpha, \mu_\alpha \rangle]\!]$, $\mathrm{Obj}'_\alpha \equiv [\![\langle \alpha, S'_\alpha, \sigma'_\alpha, \mu_\alpha \rangle]\!]$, and similarly for β.

Proof: By simultaneous induction on the depth of inference of the premises.

Suppose the premises are inferred by (WAIT) and (RETURN2) respectively so that $S_\alpha \equiv \mathsf{wait}(\beta)$, $S'_\alpha \equiv \gamma$ and $\sigma'_\alpha = \sigma_\alpha$, and $S_\beta \equiv \gamma \Rightarrow \alpha$, $S'_\beta \equiv \mathsf{nil}$, and $\sigma'_\beta \equiv \sigma_\beta$. Setting $v = l'(c)$ and $w = \overline{l'}(c)$, $[\![\mathsf{wait}(\beta)]\!] \stackrel{v}{\Longrightarrow} [\![\gamma]\!]$ and $[\![\gamma \Rightarrow \alpha]\!] \stackrel{w}{\Longrightarrow} [\![\mathsf{nil}]\!]$ and so the result follows.

The remaining cases follow by similar inductive arguments to those seen earlier. □

This completes the proof of the Claim and hence of the first part of the theorem.

We now turn to the proof of the second part of the theorem. The following lemma is sufficient to give the result.

Lemma 15 Suppose $[\![\langle \Phi^0, \Gamma^0 \rangle]\!] \Longrightarrow P$. Then

1. P is of the form

$$\begin{array}{l} \langle \tilde{k}, \tilde{b} \rangle\ (\mathrm{Classes}(\tilde{k}, \tilde{b}) \\ \quad \mid \langle \tilde{a}, \tilde{c}, \tilde{l}, \tilde{m}, \tilde{t}, \tilde{f} \rangle\ (\mathrm{Objects}(\tilde{a}, \tilde{k}, \tilde{c}, \tilde{b}, \tilde{l}, \tilde{m}, \tilde{t}, \tilde{f}) \\ \qquad\qquad \mid \mathrm{PreObjects}(\tilde{l}, \tilde{k}, \tilde{c}) \\ \qquad\qquad \mid \mathrm{Booleans}(bk, \tilde{l}, \tilde{m}, \tilde{b}, \tilde{t}, \tilde{f}))) \end{array}$$

   where

   (a) Objects is a composition of agents each of which is a derivative of an agent of the form $\mathrm{NewObject}(a, \tilde{k}, \tilde{c})$,

(b) PreObjects is a composition of agents each of which is an immediate derivative $\overline{l}\langle a\rangle.\,\mathrm{NewObject}(a,\tilde{k},\tilde{c})$ of an agent of the form $[\![Cdec]\!]$, and

(c) Booleans is a composition of agents each of which is a proper derivative of BoolClass, BoolBody, $\mathrm{BoolVal}_t$, $\mathrm{BoolVal}_f$, BoolNot or BoolAnd.

Moreover, corresponding to each component $\overline{l}\langle a\rangle.\,\mathrm{NewObject}$ of PreObjects there is a unique component of Objects in which the name l occurs free, and that component is $\equiv l(a).\,[\![\langle \alpha, S, \sigma, \mu\rangle]\!]$ for some α, S, σ, μ. Also, each component of Booleans shares a name with exactly one component of Objects, Classes or Booleans and the two agents may interact using this name.

2. For each component Q of Objects let $\mathrm{NewObject}_Q$ be the agent of which Q is a derivative in $[\![\langle \Phi^0, \Gamma^0\rangle]\!] \Longrightarrow P$. Suppose $\mathrm{NewObject}_Q \equiv [\![\langle \alpha_Q, S_Q^0, \sigma_Q^0, \mu_Q\rangle]\!]$ and $\mathrm{NewObject}_Q \xrightarrow{v_Q} Q$ is the subderivation of $[\![\langle \Phi^0, \Gamma^0\rangle]\!] \Longrightarrow P$ involving $\mathrm{NewObject}_Q$. Then there are unique S_Q, σ_Q, v_Q' such that $\mathrm{NewObject}_Q \xrightarrow{v_Q'} [\![\langle \alpha_Q, S_Q, \sigma_Q, \mu_Q\rangle]\!]$ is an initial part of the derivation $\mathrm{NewObject}_Q \xrightarrow{v_Q} Q$ and in the remainder $[\![\langle \alpha_Q, S_Q, \sigma_Q, \mu_Q\rangle]\!] \xrightarrow{v_Q''} Q$ of the derivation, no agent but the first is of the form $[\![\langle \alpha_Q, S, \sigma, \mu_Q\rangle]\!]$. In addition, for some sequence $\tilde{\ell}_Q$, $\langle \alpha_Q, \mu_Q\rangle \vdash \langle S_Q^0, \sigma_Q^0\rangle \overset{\tilde{\ell}_Q}{\longmapsto} \langle S_Q, \sigma_Q\rangle$, and if $v_Q'' \neq \varepsilon$ there are unique S_Q', σ_Q', w_Q of minimal length, and ℓ_Q such that $Q \xrightarrow{w_Q} [\![\langle \alpha_Q, S_Q', \sigma_Q', \mu_Q\rangle]\!]$ as a subderivation of a derivation of P and $\langle \alpha_Q, \mu_Q\rangle \vdash \langle S_Q, \sigma_Q\rangle \longmapsto^* \overset{\ell_Q}{\longmapsto} \langle S_Q', \sigma_Q'\rangle$.

3. There is a unique configuration $\langle \Phi, \Gamma\rangle$ such that the individual computations in the local transition system described in 2 can be combined to give a computation $\langle \Phi^0, \Gamma^0\rangle \longrightarrow^* \langle \Phi, \Gamma\rangle$ such that $P \Longrightarrow\sim [\![\langle \Phi, \Gamma\rangle]\!]$ by a sequence of transitions of minimal length.

Proof: The three assertions are proved together by induction on the number of $\xrightarrow{\tau}$ transitions in the derivation $[\![\langle \Phi^0, \Gamma^0\rangle]\!] \Longrightarrow P$. If the derivation is empty then 1, 2 and 3 all hold. For

$$P \equiv [\![\langle \Phi^0, \Gamma^0\rangle]\!] \equiv \langle \tilde{k}, \tilde{b}\rangle(\mathrm{Classes}(\tilde{k}, \tilde{b}) \mid \langle a, \tilde{c}\rangle \mathrm{NewObject}(a, \tilde{k}, \tilde{c}))$$

so PreObjects and Booleans are empty compositions and the single component of Objects is the translation of the root object.

Suppose $[\![\langle \Phi^0, \Gamma^0\rangle]\!] \Longrightarrow \overline{P} \xrightarrow{\tau} P$ and the result holds for $\overline{P}$. From the form of $\overline{P}$ it follows that the transition $\overline{P} \xrightarrow{\tau} P$ is derived from one of the following:

1. An action of a single component of Objects.

2. An interaction between two components of Objects.

3. An interaction between a component of Objects and a component of PreObjects.

4. An interaction between a component of Objects and a component of Classes.

5. An interaction between a component of Objects and a component of Booleans.

6. An interaction between two components of Booleans.

7. An interaction between a component of Booleans and a component of Classes.

By considering the possible interactions in detail it follows that P is of the same form: In cases 1 and 2 this is immediate. In case 3, a component of PreObjects in $\overline{P}$ becomes a component of Objects in P. In case 4, either PreObjects or Booleans gains a new component in P. In cases 5, 6 and 7, the form is unchanged though the components of Booleans in P may differ from those in $\overline{P}$. That the remaining assertions in 1 continue to hold follows also by detailed examination of the possible interactions.

To see that 2 and 3 hold for P first note that if Q is a component of Objects in $\overline{P}$ and Q does not participate in the transition $\overline{P} \xrightarrow{\tau} P$ then the induction hypothesis of 2 may be applied to Q. In particular if case 6 or case 7 above applies then 2 and 3 of the lemma hold for P. An important observation is that the continuation $Q \xrightarrow{w_Q} [\![\langle \alpha_Q, S'_Q, \sigma'_Q, \mu_Q \rangle]\!]$ of the derivation of NewObject_Q possible as a subderivation from $\overline{P}$ is also possible as a subderivation from P. This will become clear in the remainder of the proof.

It remains to consider the one or two components of Objects in $\overline{P}$ which do participate in the transition $\overline{P} \xrightarrow{\tau} P$. Let $\overline{Q}$ be such a component and suppose that $\overline{Q} \xrightarrow{\alpha} Q$ in $\overline{P} \xrightarrow{\tau} P$.

Suppose first that $\overline{Q}$ is not of the form $[\![\langle \alpha_Q, S, \sigma, \mu_Q \rangle]\!]$. Then by induction hypothesis

$$\text{NewObject}_Q \xrightarrow{v'_Q} [\![\langle \alpha_Q, S_Q, \sigma_Q, \mu_Q \rangle]\!] \xrightarrow{v''_Q} \overline{Q} \xrightarrow{w_Q} [\![\langle \alpha_Q, S'_Q, \sigma'_Q, \mu_Q \rangle]\!]$$

where $\langle \alpha_Q, \mu_Q \rangle \vdash \langle S^0_Q, \sigma^0_Q \rangle \overset{\tilde{\iota}_Q}{\longmapsto} \langle S_Q, \sigma_Q \rangle \longmapsto^* \overset{\iota_Q}{\longmapsto} \langle S'_Q, \sigma'_Q \rangle$. Moreover, $\overline{Q} \xrightarrow{\alpha} Q$ must be the first step of the derivation $\overline{Q} \xrightarrow{w_Q} [\![\langle \alpha_Q, S'_Q, \sigma'_Q, \mu_Q \rangle]\!]$ by virtue of the uniqueness of the continuation as a subderivation of a derivation of $\overline{P}$ guaranteed by the induction hypothesis of 2.

So it remains only to consider the case when $\overline{Q}$ is of the form $[\![\langle \alpha_Q, S, \sigma, \mu_Q \rangle]\!]$.

Lemma 16 Suppose $\overline{Q} \equiv [\![\langle \alpha, S, \sigma, \mu \rangle]\!]$ is a component of $\overline{P}$ and in the transition $\overline{P} \longrightarrow P$, $\overline{Q} \xrightarrow{\alpha} Q$. Then there are unique S', σ', w of minimal length, and ℓ such that $Q \xrightarrow{w} [\![\langle \alpha, S', \sigma', \mu \rangle]\!]$ as a subderivation of a derivation of P and $\langle \alpha_Q, \mu_Q \rangle \vdash \langle S, \sigma \rangle \longmapsto^* \overset{\ell}{\longmapsto} \langle S', \sigma' \rangle$.

Proof: By induction on the structure of S.

If $S \equiv \mathsf{true}$ then $\overline{Q}$ interacts with BoolClass and $\alpha = \overline{bk}\langle l', t, f\rangle$. The result holds with $S' \equiv \mathsf{tt}$, $\sigma' = \sigma$, $w = \bar{t}.\, l'(b)$ and ℓ empty.

The case $S \equiv \mathsf{false}$ is similar.

If $S \equiv \mathsf{new}(B)$ then $\overline{Q}$ interacts with some $[\![Cdec]\!]$ and $\alpha = \overline{k}\langle l'\rangle$. The result holds with $S' \equiv \beta$, $\sigma' = \sigma$, $w = l'(c)$ and $\ell = \mathrm{new}(B, \beta)$.

In the case $S \equiv \mathsf{self}$, no transition $\overline{Q} \xrightarrow{\alpha} Q$ is possible. Note that $\langle \alpha, \mu\rangle \vdash \langle \mathsf{self}, \sigma\rangle \longmapsto \langle \alpha, \sigma\rangle$.

If $S \equiv X$ then $\overline{Q}$ acts independently and $\alpha = \tau$. The result holds with $S' \equiv \beta$ where $\beta = \sigma(X)$, $\sigma' = \sigma$, $w = \tau^2$ and ℓ empty.

If $S \equiv \beta!M(\tilde{\beta})$ then $\overline{Q}$ interacts wih another component of Objects and $\alpha = \overline{c_0}\langle \tilde{m}\rangle$. The result holds with $S' \equiv \mathsf{wait}(\beta)$, $\sigma' = \sigma$, $w = \overline{m}(c_1, \ldots, c_n, \langle l'\rangle)$ and $\ell = \mathrm{send}(\beta, M, \tilde{\beta})$.

If $S \equiv M(\tilde{\beta})$ then $\overline{Q}$ acts independently and $\alpha = \tau$. The result holds with $S' \equiv (E, \rho)$ where $\mu(M) = \langle \tilde{Y}, E\rangle$ and $\rho = \{\tilde{Y} := \tilde{\beta}\}$, $\sigma' = \sigma$, $w = \tau^{2n+1}$ and ℓ empty.

If $S \equiv \mathit{Vdecs}\ \mathsf{in}\ E$ then $\overline{Q} \equiv [\![\langle \alpha, (E, \rho), \sigma, \mu\rangle]\!] \equiv [\![\langle \alpha, E, \sigma \oplus \rho, \mu\rangle]\!]$ where $\rho = \{\tilde{Y} := \mathsf{nil}\}$ and $\mathit{Vdecs} \equiv \mathsf{var}\ \tilde{Y}$. By induction hypothesis applied to E there are S', σ', w and ℓ such that $Q \xrightarrow{w} [\![\langle \alpha, S', \sigma', \mu\rangle]\!]$ and $\langle \alpha, \mu\rangle \vdash \langle E, \sigma \oplus \rho\rangle \longmapsto^* \xrightarrow{\ell} \langle S', \sigma'\rangle$. The result follows since $\langle \alpha, \mu\rangle \vdash \langle \mathit{Vdecs}\ \mathsf{in}\ E, \sigma\rangle \longmapsto \langle (E, \rho), \sigma\rangle \longmapsto^* \xrightarrow{\ell} \langle (S', \rho'), \sigma''\rangle$ for some ρ' and σ'' where $[\![\langle \alpha, S', \sigma', \mu\rangle]\!] \equiv [\![\langle \alpha, (S', \rho'), \sigma'', \mu\rangle]\!]$ using the transition rule (LOCAL).

If $S \equiv X := \beta$ then $\overline{Q}$ acts independently and $\alpha = \tau$. The result holds with $S' \equiv \mathsf{nil}$, $\sigma' = \sigma[X := \beta]$, $w = \tau^3$ and ℓ empty.

If $S \equiv \mathsf{answer}$ then $\overline{Q}$ interacts with another component of Objects and $\alpha = a(\tilde{m})$. The result holds with $S' \equiv (E, \rho) \Rightarrow \beta$, $\sigma' = \sigma$, $w = m(\tilde{c}, l').\tau^{2n+2}$ and $\ell = \mathrm{answer}(\beta, M, \tilde{\beta})$.

If $S \equiv \beta; S'$ then $\overline{Q}$ acts independently and $\alpha = \tau$. The result holds with $\sigma' = \sigma$, $w = \varepsilon$ and ℓ empty.

If $S \equiv \mathsf{if}\ \mathsf{tt}\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2$ then $\overline{Q}$ acts independently and $\alpha = \tau$. The result holds with $S' \equiv S_1$, $\sigma' = \sigma$, $w = \overline{b}(\tilde{m}).\, \overline{m_1}\langle t, f\rangle.\, t.\, \tau$ and ℓ empty.

The case $S \equiv \mathsf{if}\ \mathsf{ff}\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2$ is similar.

If $S \equiv \mathsf{while}\ E\ \mathsf{do}\ S''$ then $\overline{Q} \sim [\![\langle \alpha, \mathsf{if}\ E\ \mathsf{then}\ (S''; S)\ \mathsf{else}\ \mathsf{nil}, \sigma, \mu\rangle]\!]$ and the action of $\overline{Q}$ derives from an action of $[\![E]\!]$. By induction hypothesis there are S', σ', w and ℓ such that $\overline{Q} \xrightarrow{w} [\![\langle \alpha, S', \sigma', \mu\rangle]\!]$ and $\langle \alpha, \mu\rangle \vdash \langle \mathsf{if}\ E\ \mathsf{then}\ (S''; S)\ \mathsf{else}\ \mathsf{nil}, \sigma\rangle \longmapsto^* \xrightarrow{\ell} \langle S', \sigma'\rangle$. Since $\langle \alpha, \mu\rangle \vdash \langle \mathsf{while}\ E\ \mathsf{do}\ S'', \sigma\rangle \longmapsto \langle \mathsf{if}\ E\ \mathsf{then}\ (S''; S)\ \mathsf{else}\ \mathsf{nil}, \sigma'\rangle$ using the transition rule (LOOP), the result follows.

If $S \equiv \mathsf{b!}\ \mathsf{not}\ ()$ then $\overline{Q}$ interacts with Bool_t or Bool_f and $\alpha = \overline{b}(\tilde{m})$. The result holds with $S' \equiv \mathsf{b}'$, $\sigma' = \sigma$, $w = \overline{m_2}\langle l'\rangle.\, l'(b')$ and ℓ empty.

The case $S \equiv \mathsf{b!}$ and (b') is similar.

If $S \equiv \mathsf{wait}(\beta)$ then $\overline{Q}$ interacts with another component of Objects and $\alpha = l'(c)$. The result holds with $S' \equiv \gamma$, $\sigma' = \sigma$, $w = \varepsilon$ and $\ell = \mathrm{result}(\beta, \gamma)$.

If $S \equiv (\gamma, \rho)$ then $\overline{Q} \equiv [\![\langle \alpha, \gamma, \sigma, \mu \rangle]\!]$ and so no transition $\overline{Q} \xrightarrow{\alpha} Q$ is possible. Note that $\langle \alpha, \mu \rangle \vdash \langle (\gamma, \rho), \sigma \rangle \longmapsto \langle \gamma, \sigma \rangle$.

If $S \equiv \gamma \Rightarrow \beta$ then $\overline{Q}$ interacts with another component of Objects and $\alpha = \overline{l'}(c)$. The result holds with $S' \equiv \mathsf{nil}$, $\sigma' = \sigma$, $w = \varepsilon$ and $\ell = \mathrm{return}(\beta, \gamma)$.

In the case $S \equiv \beta$ no transition is possible.

If $S \equiv E!M(\tilde{E})$ the transition of $\overline{Q}$ derives from a transition of $[\![E]\!]$ and by induction hypothesis there are E', σ', w and ℓ such that $([\![E]\!] \mid [\![\sigma]\!] \mid [\![\mu]\!]) \xrightarrow{w} ([\![E']\!] \mid [\![\sigma']\!] \mid [\![\mu]\!])$ and $\langle \alpha, \mu \rangle \vdash \langle E, \sigma \rangle \longmapsto^* \overset{\ell}{\longmapsto} \langle E', \sigma' \rangle$. The result holds with $S' \equiv E'!M(\tilde{E})$ since $Q \xrightarrow{w} [\![\langle \alpha, S', \sigma', \mu \rangle]\!]$ and $\langle \alpha, \mu \rangle \vdash \langle S, \sigma \rangle \longmapsto^* \overset{\ell}{\longmapsto} \langle S', \sigma' \rangle$ using the transition rule (SEND2).

The cases $S \equiv \beta!M(\tilde{\beta}, E, \tilde{E})$, $M(\tilde{\beta}, E, \tilde{E})$, $E \Rightarrow \beta$, $X := E$, if E then S_1 else S_2 and $S_1; S_2$ follow by arguments similar to that in the preceding case.

Finally the case $S = (E, \rho)$ follows by an argument similar to that in the case *Vdecs* in E.

This completes the proof of the lemma. □

It follows that 2 and 3 of the main lemma hold for P and hence that the lemma is proved. This completes the proof of the second part of the theorem. □

References

[1] P. America, J. de Bakker, J. Kok and J. Rutten, Operational Semantics of a Parallel Object-Oriented Language, in *Conference Record of the 13th Symposium on Principles of Programming Languages* (1986), 194–208.

[2] P. America, J. de Bakker, J. Kok and J. Rutten, Denotational Semantics of a Parallel Object-Oriented Language, *Information and Computation*, 83(2), 152–205 (1989).

[3] P. America, Issues in the Design of a Parallel Object-Oriented Language, *Formal Aspects of Computing*, 1(4), 366–411 (1989).

[4] P. America, Inheritance and Subtyping in a Parallel Object-Oriented Language, in *ECOOP '87: European Conference on Object-Oriented Programming*, Lecture Notes in Computer Science 276, 234–242, Springer-Verlag, 1988.

[5] P. America and J. Rutten, A Layered Semantics for a Parallel Object-Oriented Language, in *Foundations of Object-Oriented Languages*, Lecture Notes in Computer Science 489, 91–123, Springer-Verlag, 1991.

[6] R. Milner, **A Calculus of Communicating Systems**, Lecture Notes in Computer Science 92, Springer-Verlag, 1980.

[7] R. Milner, **Communication and Concurrency**, Prentice Hall, 1989.

[8] R. Milner, Functions as Processes, Research Report no. 1154, INRIA, Sophia Antipolis, February 1990. Also to appear in Journal of Mathematical Structures in Computer Science.

[9] R. Milner, Sorts and Types in the π-calculus, in *Proceedings Third Workshop on Concurrency and Compositionality, Goslar, Germany*, Springer-Verlag, to appear.

[10] R. Milner, J. Parrow and D. Walker, A Calculus of Mobile Processes, Part I, Report ECS-LFCS–89–85, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1989. Also to appear in J. Information and Computation.

[11] R. Milner, J. Parrow and D. Walker, A Calculus of Mobile Processes, Part II, Report ECS-LFCS–89–86, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1989. Also to appear in J. Information and Computation.

[12] J. Rutten, Semantic Correctness For a Parallel Object-Oriented Language, *Siam. J. Comput.* 19(2), 341–383 (1990).

[13] F. Vaandrager, A Process Algebra Semantics of POOL, *Applications of Process Algebra*, Tracts in Theoretical Computer Science 17, 173–236, Cambridge University Press, 1990.